НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«На правах рукопису»
УДК 004.91

«До захисту допущено»
Науковий керівник кафедри
_____ І.А. Дичка
«___»_____2018р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 121 Інженерія програмного забезпечення

на тему: «Модифікований I-Match метод виявлення нечітких дублікатів в текстових даних»

Виконав:
студент VI курсу, групи КП-61м
Пастушенко Андрій Сергійович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Т.М. _____

Рецензент:
Доцент кафедри ММСА ІПСА, к.т.н., доцент,
Дідковська М.В. _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2018 року

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Лексикон — таблиця показників IDF, що обчислена на достатньо великому корпусі вхідних документів на початковій стадії роботи методу.

IDF (Inverse document Frequency) — інверсія частоти, з якою слово зустічається в документах корпусу.

$$idf(t, D) = log \frac{|D|}{|\{d \in D : t \in d\}|} \quad (1)$$

Відбиток документа — значення хеш-функції, що отримується після знаходження значень IDF у лексиконі для вхідного документа.

Хеш-функція — функція, що дозволяє перетворити дані довільної довжини в дані фіксованої довжини.

NLP – natural language processing.

SGML - Standard generalized markup language.

IDE – integrated Development Environment.

ООД – об’єктно-орієнтовний дизайн.

ВСТУП

Проблема виявлення нечітких дублікатів природномоних текстових даних є однією з найважливіших і найважчих задач аналізу документів. Актуальність цієї проблеми визначається різноманітністю додатків, у яких потрібно враховувати «подібність», наприклад, текстових документів — це й поліпшення якості пошукового індексу та архівів пошукових систем за рахунок видалення надлишкової інформації, і об'єднання статей у сюжети за подібністю їх по змісту, і фільтрація поштового й пошукового спаму, і визначення порушень авторських прав при незаконному копіюванні накових статей і тд.

Основною перешкодою для успішного розв'язку цієї задачі є гігантський обсяг даних, що містяться у базах даних сучасних пошукових машин. Такий обсяг робить неможливим «пряме» її вирішення шляхом попарного порівняння текстів документів за прийнятний час. Тому останнім часом велика увага приділяється розробці методів зниження обчислювальної складності алгоритмів пошуку дублікатів за рахунок зменшення обсягу документу до «змістового» мінімуму: попередня обробка тексту (видалення html-тегів, розділових знаків, прийменників, сполучників і т. д.), хешування певного фіксованого набору «вагомих» слів або речень документа та інші методи.

Додатковими проблемами виявлення нечітких дублікатів документів є множини текстів з невеликими змінами відносно вихідного документа, зокрема зміни слів або фраз на синонімічні аналоги, та короткі тексти, у яких важко виділити «змістову» частину та точно визначити, яка частина документу є дублікатом, а яка – ні. Тому, розробка методу, який би був стійких до незначних змін вхідних документів, є доволі актуальною задачею.

РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

1.1 Огляд існуючих методів визначення нечітких дублікатів.

1.1.1 Коефіцієнт асоціативності Джаккарда

Спочатку використовувався в математичній статистиці в якості міри схожості між об'єктами, описаними бінарними ознаками. Він заснований на аналізі співвідношення співпадаючих і ознак у різних об'єктів. Коефіцієнт Джаккарда обчислюється за такою формулою $J = \frac{A}{A+B+C}$, де A - число термінів, присутніх в обох документах, B - число термінів, присутніх в першому документі і відсутніх у другому, C - число термінів, присутніх у другому документі і відсутніх в першому.

Це перший відомий коефіцієнт подібності. Прізвище автора коефіцієнта в літературі перекладалася як: Жаккард, Джаккард. Коефіцієнт Жаккара в різних модифікаціях і записах активно використовується в екології, геоботаніці, молекулярній біології, біоінформатиці, геноміці, протеїноміці, інформатиці та ін. напрямках. Міра Жаккара еквівалентна (пов'язані однією монотонно зростаючою залежністю) мірі Серенсена і міру Сокала-Сніта для скінченних множин. В області комп’ютерної лінгвістики використовується для визначення плагіату. Для ефективного обчислення оцінки значення коефіцієнта Жаккара використовують алгоритм MinHash.

1.1.2 Метод Шинглів

Основна ідея алгоритму шинглів полягає в розбитті тексту на послідовності слів однакової довжини – шингли (від англ. shingle – черепиця). Важливою особливістю даного алгоритму є те, що шингли виділяються не один за одним, а накладаються для запобігання втрати інформації. Нижче наведено приклад розбиття фрази на шингли довжиною в чотири слова:

Вихідна фраза: *просуваючись вперед наука невпинно перекреслює сама себе*

1-й шингл: *просуваючись вперед наука невпинно*

2-й шингл: *вперед наука невпинно перекреслює*

3-й шингл: *наука невпинно перекреслює сама*

4-й шингл*: невпинно перекреслює сама себе*

Реалізація класичного алгоритму шинглів передбачає кілька основних етапів:

1. Канонізація тексту

На початковому етапі вхідні тексти приводяться до канонічного вигляду. Тобто з них видаляються усі знаки пунктуації, елементи форматування, зайві пробіли, HTML теги та стоп-слова (сполучники, частки, займенники, вигуки і т.д.), а також всі слова приводяться до початкової форми.

2. Розбиття тексту на шингли

Даний етап передбачає розбиття канонізованого тексту на підрядки однакової довжини (шингли). Найчастіше в якості кроку обираються символи або слова і значно рідше - речення. Шингли повинні виділятися з тексту не один за одним, а накладатися. Мінімальна відстань (d), між двома сусідніми шинглами дорівнює 1 слово/символ і гарантує, що при розбитті тексту на шингли не буде жодних втрат інформації.

3. Знаходження контрольних сум шинглів

Після того, як отримано множину шинглів для текстового документу, необхідно обчислити їх контрольні суми. Для цього використовують хеш-функції: md5, crc, sha тощо.

4. Побудова образу документу

В класичному алгоритмі шинглів процес побудови образу передбачає відбір лише тих контрольних сум, які будуть використовуватися при порівнянні двох документів. Звичайно це фіксована за розміром вибірка, яка містить деякі, відібрані за певним

критерієм, шингли(наприклад задану кількість мінімальних по значенню контрольних сум). У випадку використання алгоритму шинглів для класифікації текстів, кожний шингл має брати участь у порівнянні, тому в образ документу потрапляють всі без виключення шингли.

5. Пошук однакових під-послідовностей

На останньому етапі відбувається порівняння двох образів шляхом визначення ступеню їх схожості і входження.

1.1.3 Метод Winnowing

Суть даного методу полягає в тому, щоб порівнювати ні тексти документів, а набір значень хеш-функцій, що характеризують ці документів. Даний метод нагадує метод шинглів, але головна відмінність Winnowing полягає в тому, що для опису вихідного документа використовується не весь набір значень хеш-функцій (як в методі шинглів), з яких складається текст, а тільки його мала частина.

Логічно можна розділити розглянутий метод на два етапи: попередній і основний.

На попередньому етапі з документа видаляються всі неінформативні ознаки: займенники, прийменники, сполучники і т. Д. Далі розглядається документ перетворюється в один рядок шляхом видалення пробілів:

A do run run run, a do run run

Adorunrunrunadorunrun

На основному етапі перетворений документ розбивається на k грами, де параметр k задається користувачем. Для прикладу розіб'ємо перетворений вище текст на 5-грам. В результаті отримаємо:

adoru dorun orunr runru unrun nrunr runru

unrun nruna runad unado nador adoru dorun

orunr runru unrun

Потім для кожної і й k-грами обчислюється хеш-фунция. Нижче наведено приклад хешірованних k-грам:

77 74 42 17 98 50 17 98 8 88 67 39 77 74 42

17 98

Далі набір хешів k-грам розбивається на так звані вікна розміром (t-k + 1). t -шумовой поріг, тобто мінімальна довжина підрядка, при якій загальні підрядка не ігнорує. Нижче наведено приклад розбиття на «вікна» довжиною 4:

(77, 74, 42, 17) (74, 42, 17, 98)

(42, 17, 98, 50) (17, 98, 50, 17)

(98, 50, 17, 98) (50, 17, 98, 8)

(17, 98, 8, 88) (98, 8, 88, 67)

(8, 88, 67, 39) (88, 67, 39, 77)

(67, 39, 77, 74) (39, 77, 74, 42)

(77, 74, 42, 17) (74, 42, 17, 98)

З кожного j го вікна вибирається мінімальне значення хеш-функції. Якщо в сусідніх вікнах мінімальні значення хеш однакові, в результуючий набір потрапляє тільки одне значення. Отримуємо:

17 8 39 17

На заключному етапі здійснюється порівняння отриманих наборів значень хеш-функції кожного документа. В рамах даної роботи порівняння здійснювалося за допомогою коефіцієнта асоціативності Джаккарда.

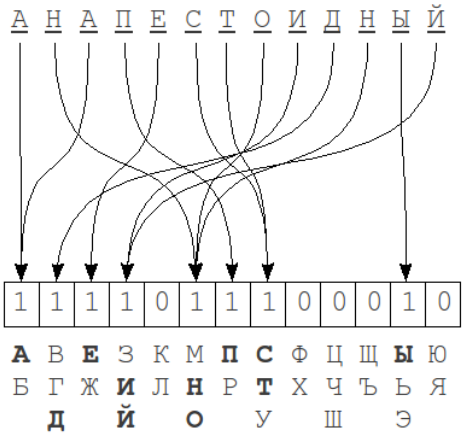
Даний алгоритм має високу швидкість роботи і гарантує, що якщо у двох порівнюваних текстових документів є загальна подстрока довжиною не менше t, вона буде знайдена.

1.1.4 Метод хешування по сигнатурі

Цей алгоритм описаний в статті Бойцова Л.М. «Хешування по сигнатурі». Він базується на досить очевидному поданні «структури» слова у вигляді бітових розрядів, яку використовують як хешу (сигнатури) в хеш-таблиці.

При індексації такі хеші обчислюються для кожного зі слів, і в таблицю заноситься відповідність списку словникових слів цього хешу. Потім, під час пошуку, для запиту обчислюється хеш і перебираються всі сусідні хеші, що відрізняються від вихідного не більше ніж в k бітах. Для кожного з таких хешів проводиться перебір списку відповідних йому слів.

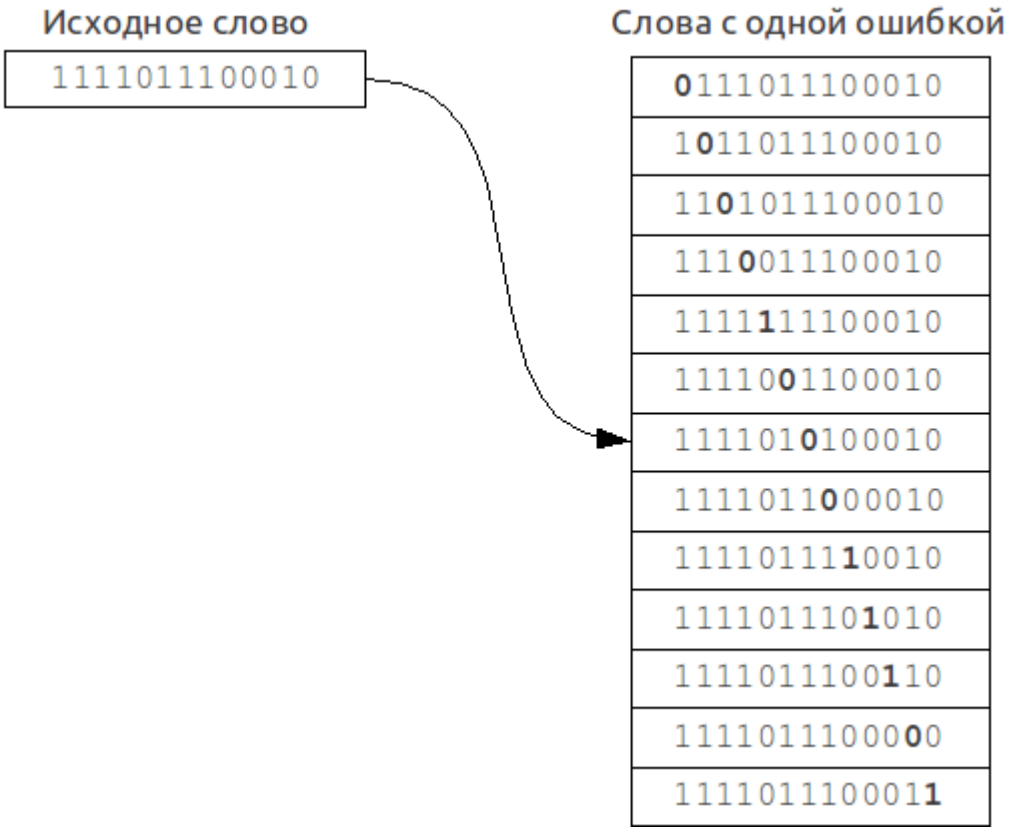
Процес обчислення хешу - кожному біту хешу зіставляється група символів з алфавіту. Біт 1 на позиції і в хеші означає, що в початковому слові присутній символ з і-ої групи алфавіту. Порядок букв в слові абсолютно ніякого значення не має.



Хеш	Список слов
0000000000000	-
...	
1111011100001	АВТОПРЕДПРИЯТИЕ, БЕЗОТРАДНАЯ, ВЕТЕРИНАРИЯ, ...
1111011100010	ПРЕВРАТНЫЙ, БЕЗРАССУДНЫЙ, АНАПЕСТОИДНЫЙ, ...
1111011100011	СОРИЕНТИРОВАТЬСЯ, БЕСПРЕПЯТСТВЕННЫЙ, ...
...	
1111111111111	ЛЕГКОИСЧЕРПЫВАЮЩИХСЯ, ВЫСОКОРАЗРЕШАЮЩИХ, ...

Видалення одного символу або не змінить значення хешу (якщо в слові ще залишилися символи з тієї ж групи алфавіту), або ж відповідний

цій групі біт зміниться в 0. При вставці, аналогічним чином або один біт встане в 1, або ніяких змін не буде. При заміні символів все трохи складніше – хеш може або зовсім залишитися незмінним, або ж зміниться в 1 або 2 позиціях. При перестановках ніяких змін і зовсім не відбувається, тому що порядок символів при побудові хешу, як і було відмічено раніше, не враховується. Таким чином, для повного покриття k помилок потрібно змінювати не менше 2k біт в хеші.



Час роботи, при *k* «неповних» (вставки, видалення і траспозиції) помилках:

$$O(|H|^k \cdot \frac{n}{2^{|H|}})$$

Недоліки:

- Через те, що при заміні одного символу можуть змінюватися відразу два біта, алгоритм, який реалізує, наприклад, спотворення не більше 2 бітів одночасно в насправді не видаватиме повного об'єму результатів через відсутність значної (залежить від відношення розміру хешу до алфавітом) частини слів з двома замінами (і чим більше розмір хешу,

тим частіше заміна символу буде приводити до спотворення відразу двох біт, і тим менше повним буде результат).

- алгоритм не дозволяє проводити префіксний пошук.

1.1.5 Метод Opt Freq

Алгоритм реалізує метод «оптимальної пошукової частоти», запропонований М. Масловим (Яндекс), і використовується для пошуку схожих документів в широкому спектрі додатків, від веб-поіка до кластеризації новин. Суть його полягає в наступному. Замість класичної метрики TF * IDF пропонується її модифікований варіант. Вводиться евристичне поняття «оптимальної частоти» для слова рівне - $ln\frac{10}{1000000} = 11.5$, тобто «Оптимальним» вважається входження слова в 10 документів з 1000000. Якщо реальне значення IDF менше «оптимального», то воно трохи (за законом параболи) підвищується до $ln\frac{10}{1000000} = 11.5$ а якщо більше, то істотно (як гіпербола) знижується до $IDF_{opt} = \frac{11.5}{IDF}$

1.1.6 TF*RIDF

Основна ідея RIDF (Residual IDF) [15] полягає в порівнянні двох способів підрахунку кількості інформації (в сенсі визначення К. Шеннона), що міститься в повідомленні про те, що дане слово входить в деякий документ (щонайменше один раз). Перший спосіб, статистичний, це звичайний

$$IDF = -log\frac{df}{N}$$

Другий спосіб, теоретичний, заснований на моделі розподілу Пуассона, яка передбачає, що слова в колекції документів розподіляються випадковим і незалежним чином, рівномірно розсіюючись з деякою

середньою щільністю. В цьому випадку відповідну кількість інформації дорівнює

$$P_{IDF} = -log\left(1 - e^{\left(\frac{-cf}{N}\right)}\right).$$

тоді,

$$RIDF = IDF - P_{IDF} = -log\left(\frac{df}{N}\right) + log\left(1 - exp\left(\frac{-cf}{N}\right)\right)$$

показує приріст інформації, що міститься в реальному розподілі слова в колекції по порівняно з рівномірно випадковим пуассонівським, тобто цінність слова. Іншими словами «хороші» (значущі, осмислені) слова повинні бути розподілені нерівномірно серед відносно невеликого числа документів (Володіти «рідкістю»), а «погані» (Беззмістовні) будуть рівномірно розсіяні по всій колекції (зустрічатися, що називається, «на кожному кроці »).

Практична реалізація. По всій колекції будується словник, що ставить кожному слову в відповідність число документів, в яких воно зустрічається хоча б один раз (df) і визначається сумарна частота кожного слова в колекції (cf).

Потім будується частотний словник документа і для кожного слова обчислюється його «вага» wt по формулою:

$TF = 0.5 + 0.5 * tf/tf_{max}$

$RIDF = -log(df/N) + log\big(1 - exp(-cf/N)\big)$

$wt = TF * RIDF$

Потім вибираються і зчіплюються в алфавітному порядку в рядок 6 слів з найбільшими значеннями wt. Як сигнатури документа обчислюється контрольна сума CRC32 отриманого рядка.

1.1.7 Метод I-Match

Розглянемо інший сигнатурний підхід, заснований вже не на синтаксичних, а на лексичних принципах. Основна ідея такого підходу полягає в обчисленні дактилограмми I-Match для подання змісту документів. З цією метою спочатку для вихідної колекції документів будується словник L, який включає слова з середніми значеннями IDF, оскільки такі слова забезпечують, як правило, більш точні результати при виявленні нечітких дублікатів. Слова з великими і маленькими значеннями IDF відкидаються. Потім для кожного документа формується безліч U різних слів, що входять в нього, і визначається перетин U і словника L. Якщо розмір цього перетину більше деякого мінімального порога (Визначається експериментально), то список слів, які входять в перетин упорядковується, і для нього обчислюється I-Match сигнатура (hash-функція SHA1).

Два документа вважаються схожими, якщо у них збігаються I-Match сигнатури (має місце колізія hash-кодів). Алгоритм має високу обчислювальну ефективність, перевершує показники алгоритму шинглів. Іншою перевагою алгоритму (Також, в порівнянні з алгоритмом шинглів, запропонованим A. Broder) є його висока ефективність при порівнянні невеликих за розміром документів. Основний недолік - нестійкість до невеликих змін змісту документа. Для подолання зазначеного недоліку вихідний алгоритм був модифікований авторами статті [1], і в нього була введена можливість багаторазового випадкового перемішування основного словника. Додатково до основного словника L створюються K різних словників L1-LK, одержуваних шляхом випадкового видалення з вихідного словника деякого невеликої фіксованої частини p слів, складової порядку 30% -35% від початкового об'єму L.

Для кожного документа замість однієї обчислюється (K + 1) I-Match сигнатура за алгоритмом, описаного вище, тобто документ

представляється у вигляді вектора розмірності (K + 1), і два документа вважаються дублікатами, якщо у них збігається хоча б одна з координат.

Якщо документ піддається невеликим змінам (порядку n слів), то ймовірність того, що принаймні одна з K додаткових сигнатур залишиться незмінною, буде дорівнює:

$1 - (1 - p^n)^K (1)$

Дійсно, ймовірність того, що зміни не торкнуться якого-небудь одного словника, дорівнює pn - ймовірності, що всі зміни потраплять в віддалену частину вихідного словника. Тоді $1 - p^n$ - ймовірність, що сигнатура зміниться, а $1 - (1 - p^n)^K$ - ймовірність, що все сигнатури зміняться (оскільки додаткові словники формуються незалежно), і, отже, (1) - і є шукана ймовірність.

Рекомендованими значеннями параметрів, добре проявили себе на практиці,

є p = 0.33 і K = 10.

Згідно зі статтею [15], алгоритм показав високі результати при використанні в різних додатках веб-пошуку і фільтрації спаму.

1.2 Огляд існуючих комерційних програмних продуктів

1.2.1 Antiplagiat

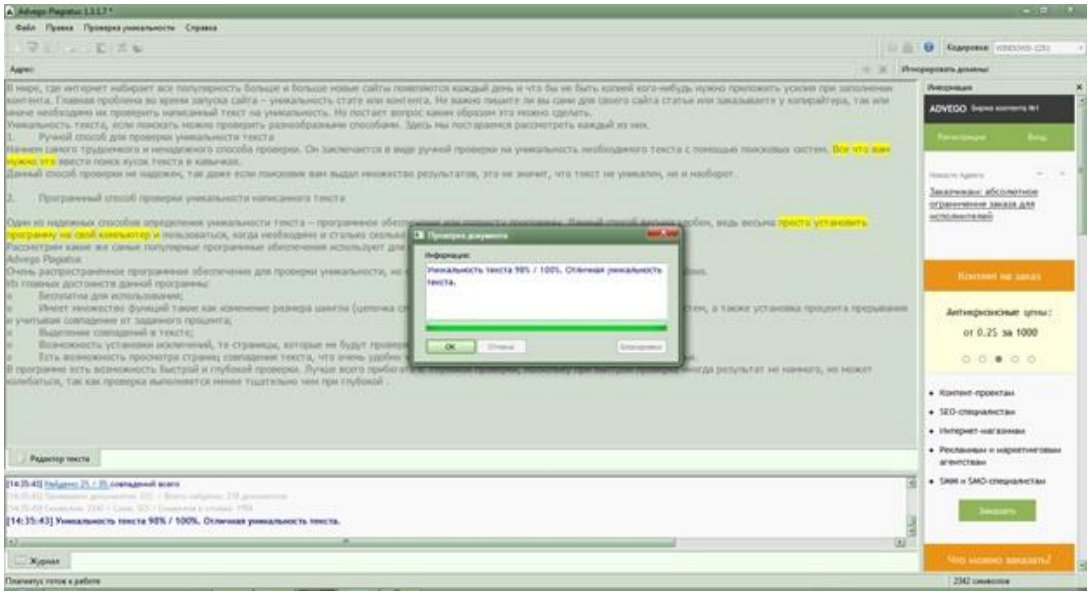
Інтернет-сервіс "Антиплагіат" пропонує набір послуг, що в сукупності реалізують технологію перевірки текстових документів на наявність запозичень із загальнодоступних джерел.

Функціональне ядро "Антиплагіат" використовує унікальні алгоритми, розроблені російськими вченими, що забезпечує швидкий і ефективний пошук запозичених фрагментів, а також гарантує коректну обробку російськомовних текстів.

Стратегічним завданням "Антиплагіат" є підвищення якості освіти в тих його частинах, де від того, хто навчається потрібно творча робота з написання рефератів, курсових і дипломних робіт та інших матеріалів власного твору. Це завдання вирішується шляхом спонукання учнів до самостійного написання текстів, а не створення їх, наприклад, шляхом компіляції знайдених в Інтернеті сторінок, що стосуються заданої тематики.

1.2.2 Advego Plagiatus

Дуже поширене програмне забезпечення для перевірки унікальності, але слід пам'ятати, що працює, воно тільки на базі ОС Windows.



З головних переваг даної програми:

- Безкоштовна для використання;
- Має безліч функцій такі як: зміна розміру шингли (ланцюжок слів), зміна використання тих чи інших пошукових систем, а також установка відсотка переривання і з огляду на збіг від заданого відсотка;
- Виділення збігів в тексті;
- Можливість установки винятків, сторінок, які не будуть перевірятися;
- Є можливість перегляду сторінок при збігу тексту, що дуже зручно, щоб дізнатися, наприклад, де знаходяться копії вашої статті.

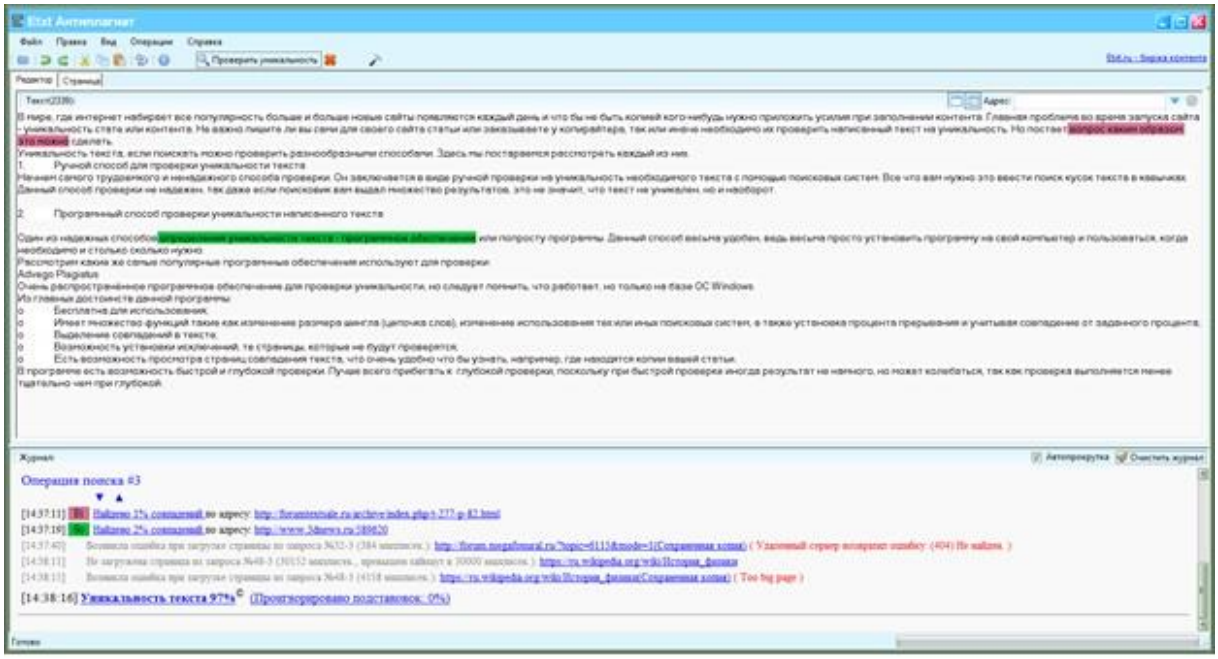
У програмі є можливість швидкої і глибокої перевірки. Краще користуватися глибокої перевіркою, тому що при швидкій перевірці іноді результат може коливатися, так як перевірка виконується менш ретельно ніж при глибокої перевірки. Перевірки виконуються досить швидко.

До недоліків використання такого програмного забезпечення можна віднести:

- немає автоматичного оновлення версії програми.
- Працює тільки під Windows

1.2.3 Etxt Антиплагиат

Etxt - головний конкурент Advego. Програмне забезпечення може бути використане на базі ОС Windows, а також є версії для Linux і Mac OS X 10.4 (і вище).



Розглянемо переваги цього програмного забезпечення:

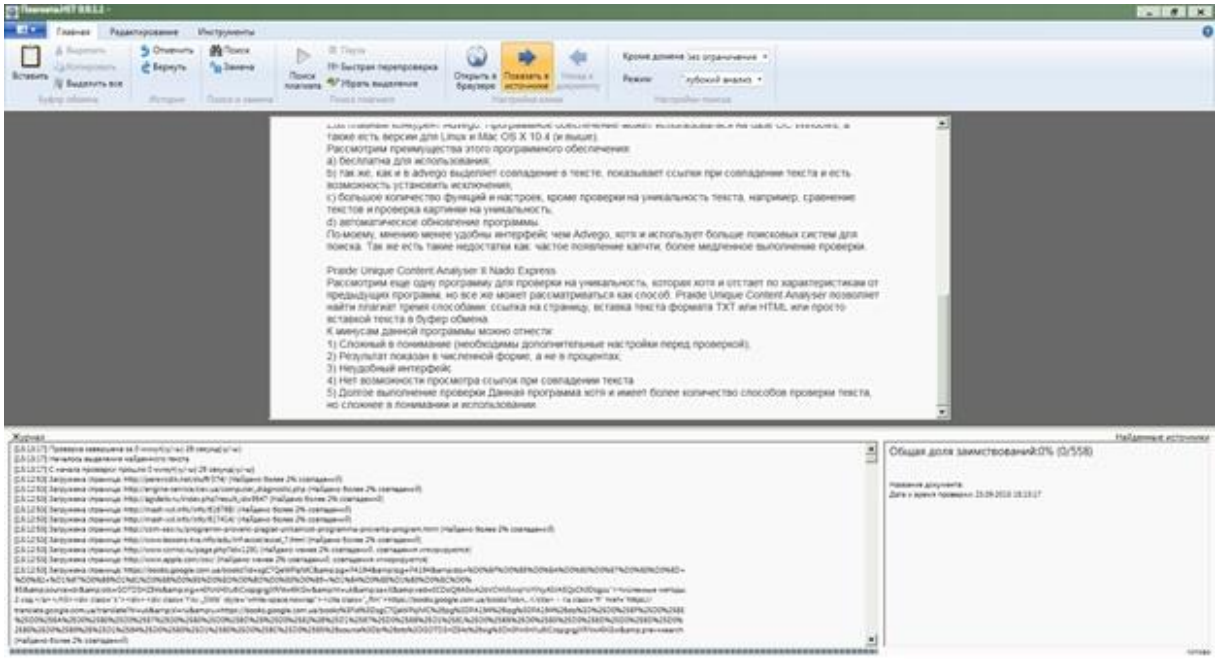
- безкоштовна для використання;
- так само, як і в Advego виділяє збіг в тексті, показує посилання при збігу тексту і є можливість встановити виключення;
- велика кількість функцій і налаштувань, крім перевірки на унікальність тексту, наприклад, порівняння текстів і перевірка унікальності картинки;
- автоматичне оновлення програми.
- використовує більше пошукових систем для пошуку.

Так само є такі недоліки як:

- часта поява капчті;
- більш повільне виконання перевірки.
- Доволі незручний інтерфейс.

1.2.4 **Плагиата.NET**

Дана програма відрізняється від інших можливістю установки програми без інсталятора (при наявності Microsoft .NET Framework 3.5 service pack).



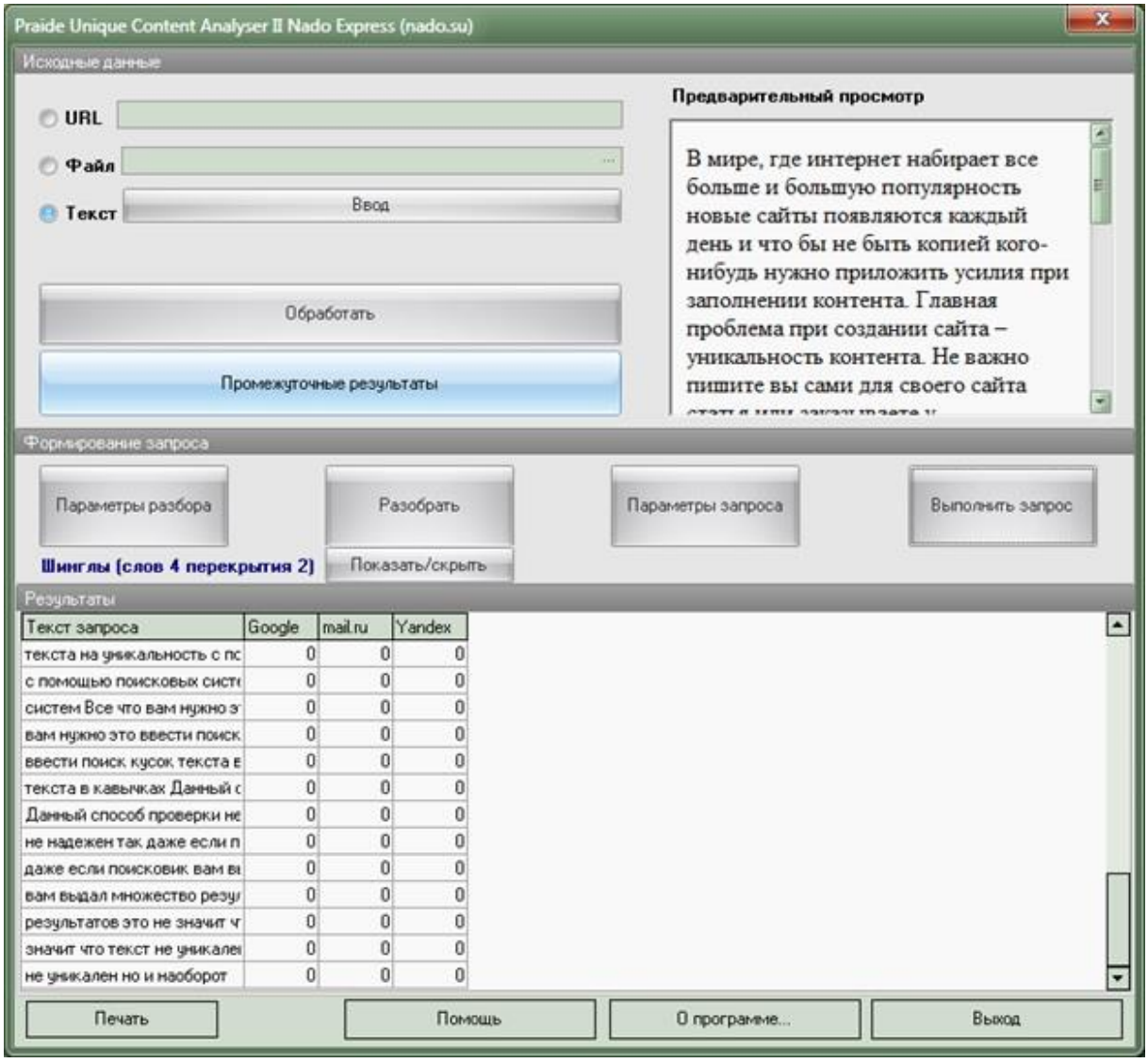
Перевагами даної програми є:

- перевірка на унікальність весь сайт;
- виділення різними кольорами знайденого відповідності слів;
- є можливість роботи з документами певного формату - rtf, doc, docx;
- додаткові функції (пошук синонімів, перевірка орфографії, заміна слів і т.д).

До недоліків можна віднести необхідність введення капчі.

1.2.5 Praide Unique Content Analyser II Nado Express

Praide Unique Content Analyser дозволяє знайти плагіат трьома способами: посилання на сторінку, вставка тексту формату TXT або HTML або просто вставкою тексту в буфер обміну.

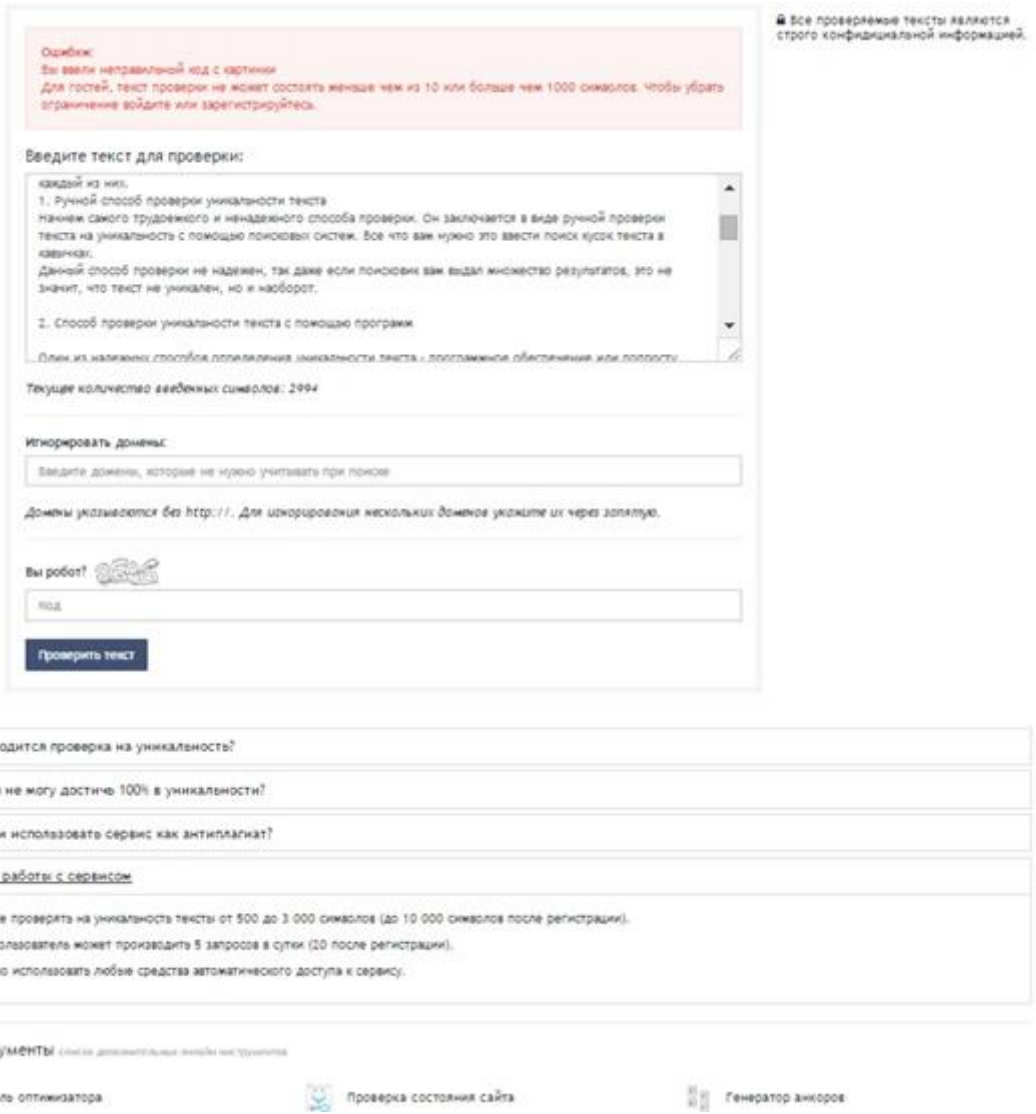


До мінусів даної програми можна віднести:

- Складний в розумінні (необхідні додаткові налаштування перед перевіркою);
- Результат показаний в чисельній формі, а не у відсотках;
- Незручний інтерфейс;
- Немає можливості перегляду посилань при збігу тексту;
- Довгий виконання перевірки.

1.2.6 pr-cy.ru

Перше чим відзначився цей сервіс - обмеження за кількістю символів, на відміну від всіх інших онлайн-сервісів, щоб перевірити нормальну статтю доведеться вже зареєструватися. Та й чи потрібно кожного разу при перевірці вводити капч, аргументується це забороною автоматичного доступу до сервісу.



Якщо ж ви зареєструєтесь, то можете виконувати до 20 запитів на добу і обсягом до 10 000 символів, на відміну від 5 раз. Ще одним подивом було саме обмеження за кількістю символів, тому що інформація про правила роботи сервісу говорить, що можна перевіряти до 3000 символів, але при перевірці ж уже вказується що до 1000 символів. Також мінусом є те, що посилання при збігу вказуються на головну сторінку, а не на саму статтю зі збігом, та й перехід по посиланню неможливий.

Хоча сервіс має ряд додаткових сервісів, а також є можливість замовити статтю на біржі нормально перевіряти тексти можна тільки після реєстрації.

1.3 Висновок

У даному розділі розглянуто існуючі теоретичні методи визначення нечітких дублікатів та наявні комерційні та безкоштовні програмні продукти, що надають дану функціональність. Так як для більшості продуктів немає в наявності інформації про реалізовані в них методи, а для деяких даний факт є комерційною таємницею, в даному розділі програмні описано програмні продукти, їх переваги та недоліки з точки зору кінцевого користувача.

Розглянувши наявні методи визначення нечітких дублікатів можна зробити висновок, що всі вони, а особливо метод хешування по сигнатурі досить чутливі до невеликих змін вхідних документів. Це означає, що методи розглядають документи у загальному вигляді і можуть не надавати достатньої точності, коли два документи різняться лиш у формулюванні фраз без зміни їх змісту.

Більшість із розглянутих методів визначення нечітких дублікатів передбачають обчислення своєрідної контрольної суми кожного з порівнюваних документів та оперують даною контрольною сумою. Однією з можливих модифікацій може бути правильний підбір хешуючої функції при обчисленні контрольної суми, що дасть змогу збільшити швидкодію та зменшити використання ресурсів процесора та пам’яті.

Розглянуті програмні продукти можна охарактеризувати наявністю привабливого інтерфейсу користувача, наявністю кількох режимів роботи, зокрема швидкого та глибокого пошуку та достатньо широку спеціалізацію (більшість з них включає в себе не тільки функціональність з визначення дублікатів документів, а й можливості пошуку синонімів, перевірки орфографії, заміни слів і т.д.

РОЗДІЛ 2 ФОРМУЛЮВАННЯ МОДИФІКАЦІЇ

2.1 Аргументація вибору базового методу

В результаті розгляду теоретичних методів визначення нечітких дублікатів виявлено, що їх можна поділити на 3 головні категорії:

- методи, що використовують шингли;
- методи, що використовують метрики відстані для обчислення подібності документів;
- сигнатурні методи.

Методи, що використовують *шингли* — підмножини послідовних термів вхідних документів порівнюють їх та визначають частину шинглів, що перетинаються. Одним із головних недоліків таких методів є необхідність у великій кількості порівнянь, а із зростанням розмірності (кількості послідовних термів, що входять до шингу) алгоритмічна складність зростає квадратично. Те ж саме відбувається при необхідності порівняння кількох (більше ніж 2-х) документів та обчислення міри їх подібності. Однією з переваг даних методів є можливість обчислення міри подібності у процентному відношенні.

Інша група методів використовує загальноприйняті метрики відстані, що використовуються в комп'ютерній лінгвістиці для порівняння термів для обчислення міри подібності документів. Загальним недоліком таких методів є недостатня стійкість та зазвичай непристосованість метрик для обробки великих об'ємів інформації.

Сигнатурний підхід передбачає в собі використання певної хеш-функції, в результаті роботи якої отримується єдине значення для обох документів, що порівнюються. Вважається, що документи є дублікатами, якшо їх "відбиток" співпадає. До переваг таких методів можна віднести використання широковживаних і часто низькорівнево оптимізованих алгоритмів хешування. Основним недоліком таких методів є неможливість обчислення міри подібності двхо документів, дані методи є досить дискретними і дозволяють відповісти тільки на запитання "Чи однакові

дані документи”, а не обчислити міру подібності двох документів у відсотках.

Методи, в основі яких лежить сигнатурний підхід, різняться між собою алгоритмами обробки документів, що порівнюються перед тим як подавати їх на вхід хеш-функції. Отже, сигнатурні методи працюють за такою схемою:

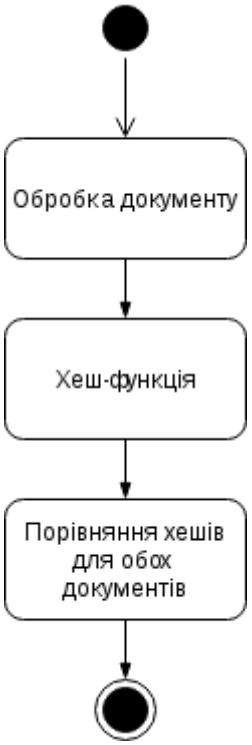


Рис 1. Схема роботи сигнатурних методів

При цьому серед усіх сигнатурних методів особливо зарекомендував себе метод I-Match, що має високу швидкодію та доволі високу точність. Високу швидкодію цього методу можна пояснити тим фактом, що даний метод потребує попереднього обчислення таблиці IDF значень на обраному корпусі документів – «лексикону». Даний «лексикон» не залежить від документів, що порівнюються, і може бути обчислений ще на етапі реалізації методу розробником.

Це дає можливості для модифікації як на етапі обчислення «лексикону», так і на етапі конкретного обчислення сигнатур документів, що порівнюються.

2.2 Можливості покращення методу I-Match

Для визначення можливостей для покращення базового методу доцільно розглянути детально базовий метод I-Match та його існуючі модифікації.

Основними етапами базового методу I-Match є:

1. Перед початком роботи методу, на підготовчому етапі, відбувається обчислення так званого “лексикону” - таблиці значень IDF, обчислених на достатньо великому корпусі документів. При цьому величина корпусу достатньо сильно впливає на точність методу, так як після обчислення пропонується відкинути значення з найбільшими і найменшими значеннями IDF для уникнення впливу на дактилограму документа частовживаних слів, сполучників і тд.
2. Для документів, що порівнюються, визначити набір унікальних термів. Порядок слів та речень у тексті не мають ніякого значення і не впливають на значення дактилограми. Також пропонується видалити знаки пунктуації, видалити стоп-слова, провести стемінг та видалити слова, що містять спецсимволи.
3. Для кожного терму у вхідних документах виконати пошук значення IDF у «лексиконі». Отримане значення IDF подати на вхід хеш-функції. В даному випадку хеш-функція працює в кумулятивному режимі, щораз накопичуючи і переобчислюючи дактилограму документа.
4. В результаті для кожного документу, що порівнюється, буде отримано значення хеш-функції — «відбитку». Якщо у порівнюваних документів ці значення збігаються — ці документи вважаються дублікатами.

У існуючій модифікації методу I-Match, що описана в статті в статті A. Kołcz та A. Chowdhury зазначається, що точність методу I-Match в основному залежть від правильності вибору лексикону L перед початком

роботи методу. Тому в даній статті запропоновано кілька модифікацій, що базуються на експерементальних результатах і спрямовані на підвищення точності шляхом модифікації алгоритму обчислення “лексикону”.

Одна з модифікацій пропонує відкидати значення з високими та низькими значеннями IDF. При цьому потрібно мати на увазі, що при відкиданні значень з великими IDF не тільки виключаються дуже рідкісні слова, але й слова з помилками, що дозволяє методу мати стійкість до невеликої кількості помилок в лексиконі. Порогові значення IDF при яких пропонується вікидати терм, повинні обиратися експерементальним шляхом і залежати від конкретного набору документів у корпусі. Зокрема, для наукових корпусів, відкидання значень з великим значенням IDF може означати відкидання термінів у наукових текстах. В художніх корпусах це може означати виключення рідкісних зворотів або авторських неологізмів.

Інша модифікація спрямована вирішити іншу проблему оригінального методу: низька ефективність порівняння, коли величина порівнюваних документів є досить великою та при цьому спостерігається велика різниця між термами документа та термами лексикону (перетин лексикону L та множини унікальних термів документу U є досить малим).

Описана модифікація пропонує ввести додатковий “вторинний лексикон”, який повинен бути набагато більшого розміру ніж основний та може бути побудований без застосування фільтрації термів. В даному випадку, коли пошук в основному лексиконі не дав результатів, виконується пошук у вторинному лексиконі. Експерементальні результати також показують, що є сенс виконувати перемішування початкового та додаткового лексиконів та цим самим нормалізувати значення IDF для різних корпусів документів.

Також розглядається можливість створення кількох лексиконів з різних корпусів документів та обчислювати кілька “відбитків” документів на різних лексиконах.

Отже, проаналізувавши базовий метод I-Match та наявні модифікації цього методу можна зробити висновок, що модифікації, націлені на збільшення точності в більшості випадків оперують над процесом обчислення початкового “лексикону” перед початком роботи алгоритму обчислення “відбитку” документа.

З іншого боку, проценс обчислення “лексикону” хоч і враховує терми, що повторюються, але використовує для порівняння термів примітивне порівняння.

Також, в оригінальній статті зазначено, що в якості хеш-функції для обчислення “відбитку” документа потрібно використовувати алгоритм SHA-1 (NIST 1995), але при цьому не надано аргументації, чому використано саме цей алгоритм. Тому, на нашу думку, доцільно провести дослідження того, як вибір хеш-фукції впливає на точність та швидкодію методу.

2.3 Модифікація методу виявлення нечітких дублікатів в тектових даних I-Match

З огляду на висновок до попереднього пункту можна сформулювати модифікацію методу I-Match:

- при обчисленні «лексикону» враховувати синонімічні колізії у вхідних документах. Видалення синонімів потребує зміни процедури порівняння термів при включенні їх в таблицю “лексикону”. Порівнння повинно проводити пошук у словнику синонімів та виключати входження синонімів у результуючу таблицю IDF. Словник синонімів потребує реалізації з використанням структур даних та алгоритмів, що дозволяють проводити швидкий пошук на великих об’ємах даних. Видалення синонімічних входжень в таблиці дозволить зменшити використання пам’яті та збільшити точність методу завдяки тому, що значення IDF для синонімів є одним значенням у таблиці “лексикону”;
- при попередній обробці документа, що порівнюється, при виділенні унікальних термів, враховувати синонімічні входження. Це дозволить зменшити обсяг множини унікальних термів для документу, а в результаті — зменшити затрати процесорного часу на обробку документу.
- перед обчисленням “відбитку” документа, при пошуку значення IDF у таблиці “лексикону” вважати синоніми — однаковими термами. Це дозволить уникнути повторень та забезпечити максимальну унікальність термів, що подаються на вхід хеш-функції.

2.4 Особливості алгоритму реалізації модифікованого методу

Отже, з огляду на запропоновану модифікацію, можна зробити висновок, що реалізація модифікованого методу визначення нечітких дублікатів потребує алгоритмів та структур даних, що забезпечують швидкий пошук на великих об’ємах даних. Задача пошуку інформації в програмній інженерії є достатньо складною і актуальною задачею, тому варто зупинитись на описі цієї задачі.

Пошук інформації являє собою процес виявлення в деякій множині документів (текстів) всіх таких, які присвячені зазначеній темі (предмету), задовольняють заздалегідь визначеним умовам пошуку (запиту) або містять необхідні (відповідно до інформаційної потреби) факти, відомості, дані.

Центральна задача ІП – допомогти користувачеві задовольнити його інформаційну потребу. Так як описати інформаційні потреби користувача технічно непросто, вони формулюються як деякий запит, який представляє з себе набір ключових слів, що характеризує те, що шукає користувач. Класична задача ІП, з якої почався розвиток цієї галузі, – це пошук документів, що задовольняють запиту, в рамках деякої статичної колекції документів.

Але список завдань ІП постійно розширюється і тепер включає:

– питання моделювання;

– класифікація документів;

– фільтрація документів;

– кластеризація документів;

– проектування архітектур пошукових систем і користувальницьких інтерфейсів;

– отримання інформації, зокрема анотування і реферування документів; – мови запитів та ін.

Процес пошуку включає послідовність операцій, спрямованих на збір, обробку та надання необхідної інформації зацікавленим особам. У загальному випадку процес пошуку інформації складається з наступних етапів:

– формулювання запиту природною мовою, вибір пошукових системи і сервісів, формалізація запиту на відповідній ІПМ;

– проведення пошуку в одній або декількох пошукових системах;

– огляд отриманих результатів (посилань);

– попередня обробка отриманих результатів: перегляд змісту посилань, вилучення та збереження релевантних і пертінентних даних;

– при необхідності, модифікація запиту і проведення повторного (уточнюючого) пошуку з подальшою обробкою отриманих результатів.

Інформаційний пошук має на увазі використання певних стратегій, методів, механізмів і засобів. Поведінка користувача, що здійснює управління процесом пошуку, визначається не тільки інформаційною потребою, а й інструментальним різноманітністю.

Стратегія пошуку – загальний план (концепція, перевага, установка) поведінки системи або користувача для вираження і задоволення інформаційної потреби користувача, обумовлений як характером мети і видом пошуку, так і системними «стратегічними» рішеннями – архітектурою БД, методами і засобами пошуку в конкретній ІПС. Вибір стратегії в загальному випадку є оптимізаційним завданням. На практиці в значній мірі він визначається мистецтвом досягнення компромісу між практичними потребами і можливостями наявних коштів. Метод пошуку – сукупність моделей і алгоритмів реалізації окремих технологічних етапів: побудови пошукового образу запиту (ПОЗ), відбору документів (зіставлення пошукових образів запитів і документів), розширення і реформулювання запиту, локалізації та оцінки видачі.

Пошуковий образ запиту – записаний на ІПМ текст, що виражає смисловий зміст інформаційного запиту і містить вказівки, необхідні для найбільш ефективного здійснення інформаційного пошуку. Методи пошуку, тобто виділення підмножини документів, які потенційно містять опис рішення задачі відбору документів, є відображенням процесу знаходження рішення і залежать від характеру завдання і предметної області.

Розглядаючи пошук як ітеративний процес, методи скорочення простору перебору (підмножини, що переглядається) утворюють по суті методологічну основу стратегії пошуку і можуть бути розділені на наступні класи – методи пошуку в:

– одному просторі (зазвичай, тематичному);

– ієрархічно упорядкованому просторі;

– альтернативних просторах;

– динамічному (змінюваному в процесі пошуку) просторі.

Метод побудови ПОЗ повинен забезпечувати ефективні способи побудови запиту для досягнення цілей різного типу. Механізми пошуку – сукупність реалізованих у системі моделей і алгоритмів процесу формування видачі документів у відповідь на пошуковий запит. Засоби пошуку, з одного боку, – взаємозалежний комплекс інформаційно-пошукових мов (ІПМ) і мов визначення/управління даними, що забезпечує структурні та семантичні перетворення об’єктів обробки (документів, словників, сукупностей результатів пошуку), а з іншого, – об’єкти користувальницького інтерфейсу, забезпечують управління послідовністю вибору операційних об’єктів конкретної ІПС.

Пошукові технології – уніфіковані (оптимізовані в рамках конкретної ІПС) послідовності ефективного використання окремих засобів пошуку в 22 процесі взаємодії користувача з системою для сталого отримання кінцевого і проміжних результатів. Навігація як реалізація

процесу пошуку за запитом в обраній БД – цілеспрямована, обумовлена стратегією, послідовність використання методів, засобів і технологій конкретної ІПС для отримання та оцінки результату. Засоби навігації дозволяють користувачеві здійснювати управління процесом пошуку. Вони надаються користувачеві у вигляді інтерфейсу, що дозволяє організувати більш-менш ефективний процес взаємодії з БД. При цьому «дружність» інтерфейсу характеризується не тільки ергономічністю і зрозумілістю, а й варіантністю вибору операційних об’єктів.

Процес пошуку інформації представляє послідовність кроків, що призводять при посередництві системи до деякого результату, і дозволяють оцінити його повноту. Так як користувач зазвичай не має вичерпних знань про інформаційне змісті ресурсу, в якому проводить пошук, то оцінити адекватність виразу запиту, так само як і повноту одержуваного результату, він може, грунтуючись лише на зовнішніх оцінках або на проміжних результатах і узагальненнях, зіставляючи їх, наприклад , з попередніми.

Типологія пошукових задач За характером і рівня співвідношення у предметі пошуку відомого і невідомого (як ступеня семантичної невизначеності) можна виділити три типи пошукових завдань.

1. Предметний (або атрибутивний) вид пошуку - пошук об’єкта, коли відомо, що цей об’єкт існує (наприклад, пошук фактографії або праць конкретного автора). Пошукова модель (логічна ідентифікація об’єкта пошуку) може бути представлена як пошук по атрибутам. Для документального пошуку – це відбір по логічному вираженню над іменами понять, що задаються термінами або їх комбінаціями.

2. Тематичний вид пошуку – підбір інформації за деякою темі, наприклад, для пошуку методу вирішення практичного завдання. Тематичний пошук – це знаходження в середовищі ІПС описів актуально існуючих в предметній області основної діяльності об’єктів, властивості яких можуть бути повністю визначені на вже відомій множині атрибутів.

Пошукова модель в цьому випадку – це пошук по частині відомого поняття або зв’язкам, частково заданим комбінацією характеристичних ознак. Тематичний пошук реалізується як послідовність атрибутивних пошуків. 3. Форма проблемного пошуку – знаходження в інформаційному середовищі описів об’єктів або їх складових, потенційно існуючих в предметній області основної діяльності і в сукупності, можливо, таких, що утворюють ціле, властивості якого будуть більше суми властивостей частин. Тобто цим властивостям в явній формі не відповідають «власні» атрибути, а нова 23 властивість, наприклад, може бути задана комбінацією вже відомих атрибутів. Логічна пошукова модель для цього випадку – пошук «схожих» документів, зміст яких деяким чином асоціюється із завданням користувача. Види пошуку Повнотекстовий пошук – пошук по всьому вмісту документа. Приклад повнотекстового пошуку – будь-яка пошукова система Інтернет, наприклад www.google.com, www.yandex.ru. Як правило, повнотекстовий пошук для прискорення пошуку використовує попередньо побудовані індекси. Найбільш поширеною технологією для індексів повнотекстового пошуку є інвертовані індекси. Пошук по метаданих – це пошук за деякими атрибутам документа, що підтримується системою – назва документа, дата створення, розмір, автор тощо. Приклад пошуку за реквізитами – діалог пошуку в файлової системі (наприклад, MS Windows). Пошук по зображенню – пошук за змістом зображення. Пошукова система розпізнає зміст фотографії (завантаженої користувачем або доданий URL зображення). У результатах пошуку користувач отримує схожі зображення. Так працюють пошукові системи Xcavator, Retrievr, PolarRose, PicollatorOnlinebyRecogmission. На сьогоднішній день існують програмні додатки та Інтернет-сервіси які здійснюють пошук в аудіо файлах по уривку. Таку послугу, зокрема, надає Google. Компоненти інформаційного пошуку Для забезпечення процедури інформаційного пошуку в ІПС можна виокремити два рівні розгляду – абстрактний і конкретний. Абстрактною ІПС прийнято вважати сукупність ІПМ, правил

індексування та критеріїв видачі, або критеріїв смислової відповідності інформації.

Під конкретною ІПС розуміється практично реалізована, що містить у своєму складі масив документів в якому проводиться пошук, технічні засоби, а також людей, що взаємодіють із системою. У відповідності до виділення в ІПС абстрактного і конкретного рівнів та з урахуванням особливостей зберігання документальної інформації (бібліотеки, архіви та інші сховища) процедуру інформаційного пошуку можна розділити на дві складові:

– семантичне осмислення запиту і видача адрес відповідних запиту документів;

– знаходження самих документів.

У символічній формі абстрактна ІПС являє собою сукупність ІПМ, правил індексування й логіки, критерії змістовної відповідності.

РОЗДІЛ 3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МЕТОДУ

3.1 Опис засобів розробки програмного забезпечення

З огляду на те, що більшість лабораторій обладнані персональними комп’ютерами з встановленою ОС Windows та інколи мають поганих Інернет-зв’язок, було вирішено реалізовувати програмний продукт у вигляді Windows-застосунку, що постачається з усіма необхідними компонентами, що необхідні для повноцінної роботи застосунку.

Пропонується розглянути найпопулярніші інструменти для створення Windows-застосунків.

1.1.1. Swing

Swing - це інструментарій віджета GUI для Java. Він є частиною класу Java Foundation Classes (JFC) - API для надання графічного інтерфейсу (GUI) для Java-програм.

Swing була розроблена, щоб забезпечити більш складний набір компонентів графічного інтерфейсу, ніж попередній інструментарій Abstraction Window (AWT). Swing забезпечує зовнішній вигляд і відчуття, що імітує зовнішній вигляд декількох платформ, а також підтримує стильний зовнішній вигляд, який дозволяє додаткам виглядати і відчувати не пов'язані з основною платформою. Він має більш потужні та гнучкі компоненти, ніж AWT. Крім знайомих компонентів, таких як кнопки, прапорці та етикетки, Swing надає кілька розширених компонентів, таких як панель вкладок, панелі прокрутки, дерева, таблиці та списки.

Переваги:

- універсальність інтерфейсу розроблюваних програм на всіх платформах та операційних системах;
- можливість для розширення: Swing – розподілена архітектура, яка підтримує додавання реалізацій користувача вказаної структури класів: користувачі можуть створити реалізацію

компонентів, щоб замінити вигляд чи поводження компонентів.

Недоліки:

- основним недоліком є повільна робота в порівнянні з нативними реалізаціями;
- не підтримується;
- застаріла архітектура;
- повільна робота
- сервісна логіка пов'язана з представленням, тому неможливо повторно використовувати логіку в інших цілях;
- зміна дизайну додатку часто вимагає від розробника змін в коді, що не дозволяє розділити обов’язки розробника і дизайнера;
- наявність обробників подій в коді негативно впливає на можливості для тестування розробленого програмного продукту.

1.1.2. Qt Framework

Qt - це крос-платформний додаток та набір інструментів та віджетів для створення класичних і вбудованих графічних користувацьких інтерфейсів, а також програм, що працюють на різних програмних та апаратних платформах, які практично не змінюють базову кодову базу, але як і раніше є нативною програмою з нативними можливостями і швидкість

Qt в даний час розробляється як компанією The Qt Company, що зареєстрована в публічній біржі, так і Qt Project під управлінням відкритого коду, залучаючи окремих розробників та фірм, які працюють над підвищенням Qt. Qt доступний як з патентованими, так і з відкритими версіями ліцензій GPL 2.0, GPL 3.0 та LGPL 3.0.

Qt використовується для розробки графічних користувацьких інтерфейсів (GUI) та багатоплатформних додатків, що працюють на всіх

основних настільних платформах та більшості мобільних або вбудованих платформ. Більшість графічних програм, створених за допомогою Qt, мають оригінальний інтерфейс, у цьому випадку Qt класифікується як набір інструментів віджетів. Також можуть бути розроблені не-графічні програми, такі як інструменти командного рядка та консолі для серверів. Прикладом такої програми, що не є графічним інтерфейсом користувача з використанням Qt, є Cutelyst web framework.

Qt підтримує різні компілятори, включаючи компілятор GCC C ++ та набір Visual Studio і має велику підтримку інтернаціоналізації. Qt також забезпечує Qt Quick, який включає декларативну мову сценаріїв, що називається QML, що дозволяє використовувати JavaScript для забезпечення логіки. За допомогою Qt Quick було можливим швидке розробка додатків для мобільних пристроїв, тоді як логіка все ще може бути написана з нативним кодом, а також для досягнення найкращої продуктивності.

Інші функції включають в себе доступ до бази даних SQL, аналіз XML, аналіз JSON, управління потоками та підтримку мережі.

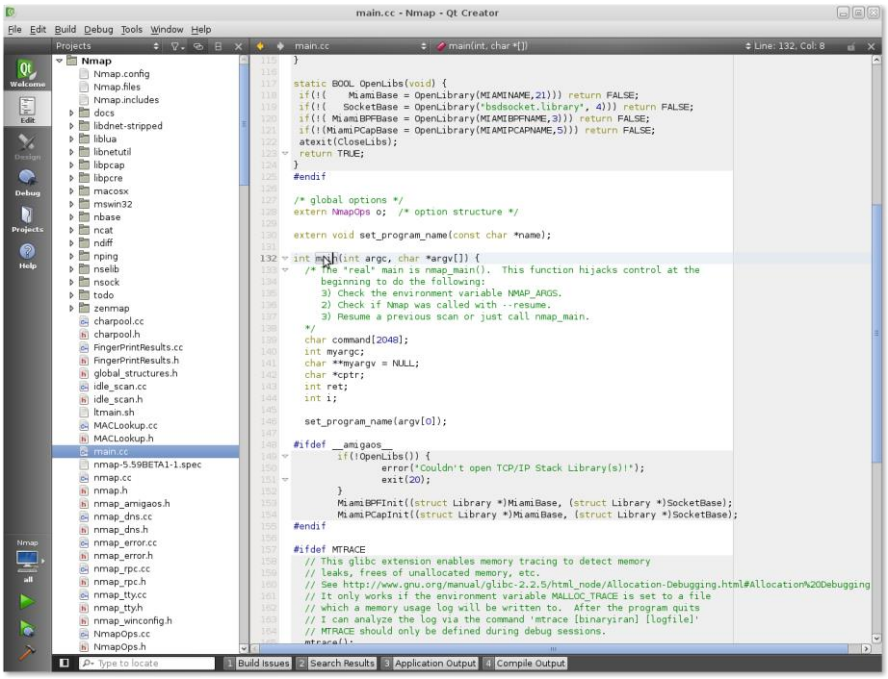


Рис. 1. QT Creator

Переваги:

- Кросплаторменність.
- Відкритий програмний код.
- Візуальний редактор .
- Використання C++ для розробки (швидкодія).

Недоліки:

- Розмір скомпільованого програмного продукту.
- Неможливість використання платформо-специфічних переваг.
- Складність опановування мовою програмування C++.
- Швидке і надлишкове розростання можливостей бібліотеки.

1.1.3. Windows Forms

Windows Forms - це технологія інтелектуальних клієнтів для .NET Framework. Вона представляє собою набір керованих бібліотек, що спрощують виконання стандартних завдань, таких як читання з файлової системи та запис в неї. При використанні середовища розробки, як Visual Studio, можна створювати інтелектуальні клієнтські додатки Windows

Forms, які відображають інформацію, запитують введення від користувачів і обмінюються даними з віддаленими комп'ютерами по мережі.

В Windows Forms форма - це візуальна поверхня, на якій виводиться інформація для користувача. Обычно додаток Windows Forms будується шляхом розміщення елементів керування формуванням і написанням коду для реагування на дії користувача, наприклад, миші або натискання клавіш. Елемент управління - це окремий елемент користувацького інтерфейсу, призначений для відображення або введення даних.

При виконанні користувачем будь-якої дії з формою або одним з її елементів управління створюється подія. Програма реагує на ці події за допомогою коду і обробляє події при їх виникненні. Подробнее см. в розділі Создание обработчиков событий в Windows Forms.

Windows Forms включає в себе широкий набір елементів управління, які можна додати на форми: текстові поля, кнопки, розкриваючі списки, перемикачі та навіть веб-сторінки. Список всіх елементів управління, які можна використовувати в формі, представлені в розділі елементи управління для використання у формах Windows Forms. Якщо існуючий елемент управління не задовольняє потреби, в Windows Forms можна створити користувацькі елементи керування за допомогою класу UserControl.

В складі Windows Forms входять багатофункціональні елементи користувацького інтерфейсу, що дозволяють відтворювати можливості таких складних додатків, як Microsoft Office. Використовуючи елементи керування ToolStrip і MenuStrip, можна створити панелі інструментів і меню, що містять текст та малюнки, підменю та інші елементи керування, такі як текстові поля та поля з списками.

Переваги:

- Нативна підтримка у Windows.
- Використання MS Visual Studio для розробки застосувань.
- Широкий набір компонентів для розробки.
- Подійно-орієнтована архітектура.

Недоліки:

- Складність створення складних інтерфейсних рішень.

1.1.4. Windows Presentation Foundation

Windows Presentation Foundation (WPF) - це Microsoft графічна підсистема для відображення користувацьких інтерфейсів у додатках Windows. WPF, раніше відомий як "Авалон", спочатку був випущений в рамках .NET Framework 3.0 у 2006 році. WPF використовує DirectX. WPF намагається забезпечити послідовну модель програмування для побудови додатків та розділяє користувальницький інтерфейс з бізнес-логіки. Він нагадує схожі XML-орієнтовані об'єктні моделі, такі як ті, що реалізовані в XUL і SVG.

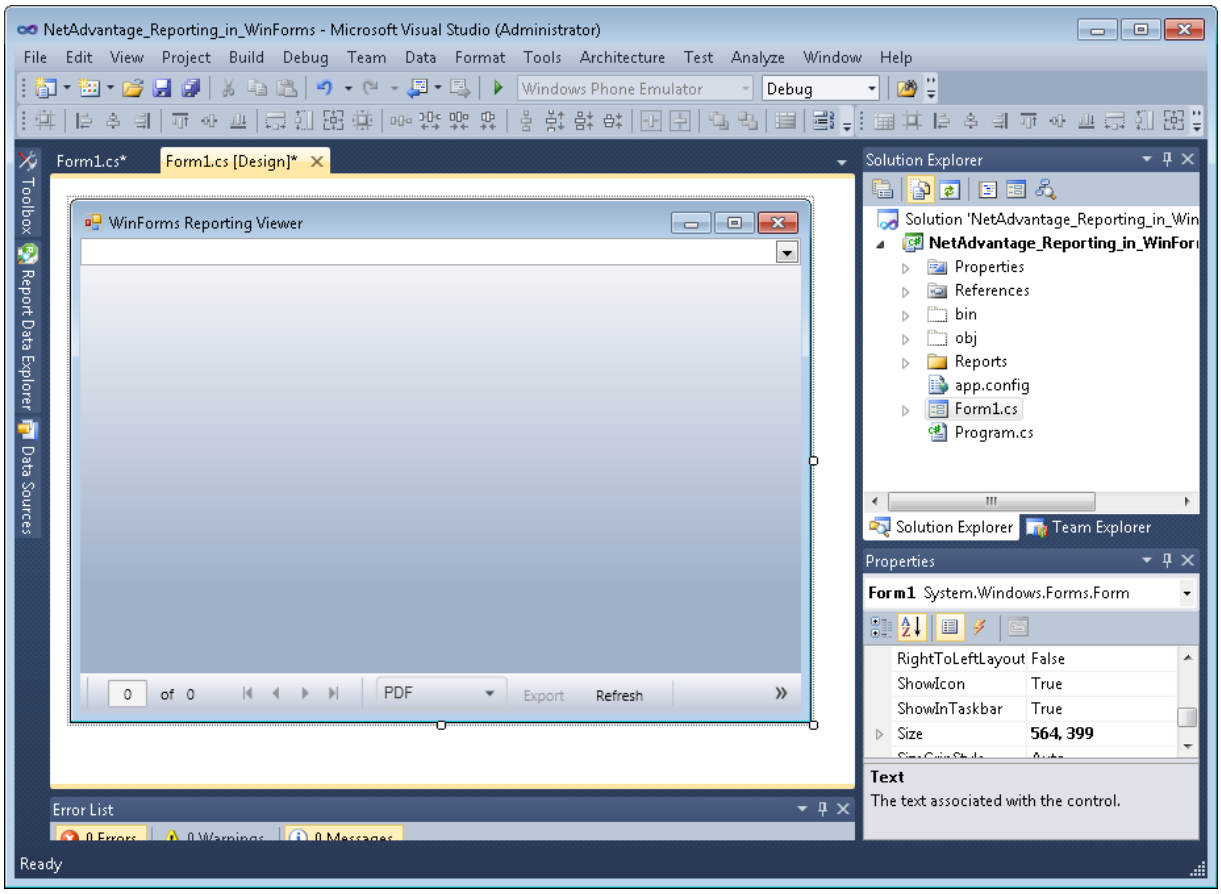


Рис. 2. Visual Studio IDE в режимі візуальної розробки

WPF використовує XAML, мову на базі XML, для визначення та поєднання різних елементів інтерфейсу [1]. Програми WPF можуть бути розгорнуті як окремі настільні програми або розміщені як вбудований об'єкт на веб-сайті. WPF має на меті об'єднати ряд загальних елементів

користувальницького інтерфейсу, таких як 2D / 3D рендеринг, фіксовані та адаптивні документи, типографіка, векторна графіка, анімація часу виконання та попередньо відтворені носії. Потім ці елементи можуть бути пов'язаними та маніпулювати на основі різних подій, взаємодії користувачів та прив'язування даних.

В якості графічної технології WPF використовує технологію DirectX, що дозволяє використовувати апаратне прискорення при відмальовуванні графічних елементів інтерфейсу користувача.

У порівнянні з попередником, Windows Forms, WPF пропонує нову командну модель, що надає дві наступних важливих можливості:

- делегування подій відповідними командам;
- підтримка включеного стану елемента керування в синхронізованому виді за допомогою стану відповідної команди.

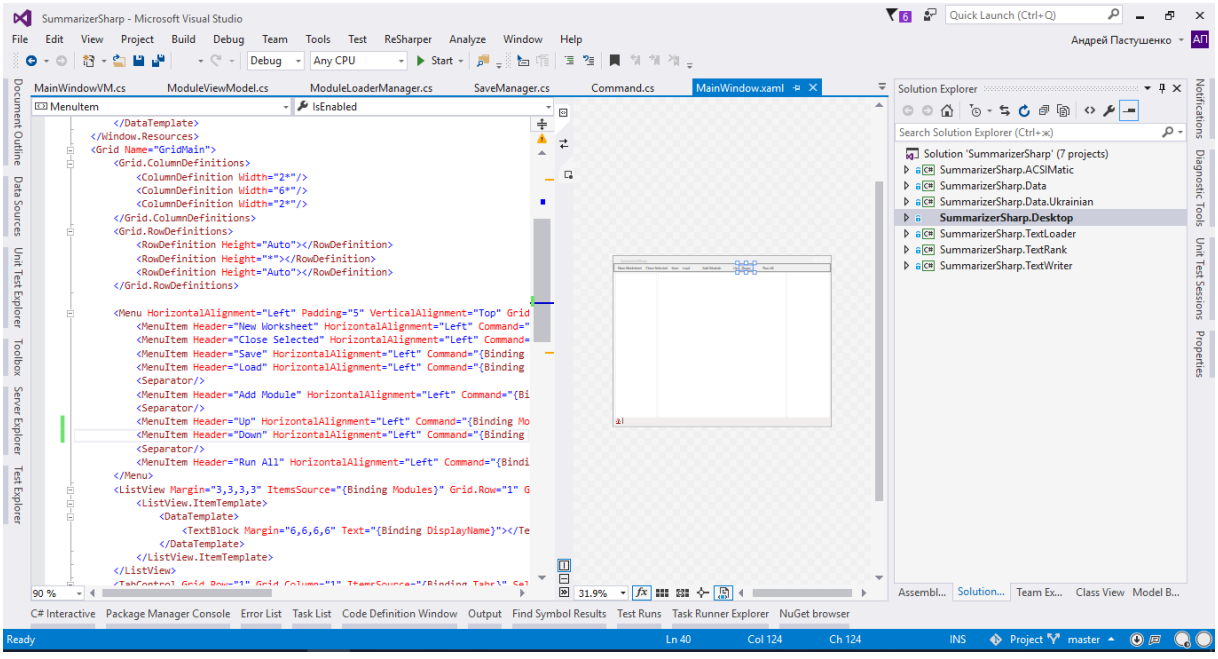


Рис. 3. Visual Studio 2015. Розробка WPF форми

Прив'язка даних – це спосіб доступу до інформації та відображення на користувальницькому інтерфейсі застосунка без написання коду, що виконує всю цю роботу. Часто «товсті» клієнти використовують двонаправлену прив'язку даних, що додає можливості – «заштовхування» інформації з користувальницького інтерфейсу назад в об'єкт загальної

логічної моделі. Оскільки багато Windows-застосунків пов'язані з даними (і всі вони в певний час мають потребу у взаємодії з даними), прив'язка даних є однією з основних технологій для побудови користувальницьких інтерфейсів, як WPF.

За прив'язку даних у WPF відповідає клас Binding.

Переваги:

- Можливість створення гнучких інтерфейсів.
- Використання шаблону MVVM.
- Використання GDI+.
- Використання для розробки MS Visual Studio.

Недоліки:

- Складність освоєння бібліотеки при переході з Windows Forms.
- Більші вимоги до апаратного забезпечення у порівнянні з Windows Forms.

За результатами дослідження було вирішено, що програмне забезпечення буде розроблюватись у вигляді Windows-застосунок з використанням бібліотеки WPF. Це надасть змогу створювати складний інтерфейс, що потрібний для конфігурування параметрів алгоритмів реферування. Додатково, використання шаблону MVVM надасть змогу розділити розробку інтерфейсу користувача (представлення) та ядра системи (моделі), що зменшить зв’язність модулів системи.

MVVM була створена з метою поділу праці дизайнера і програміста, що є неможливим, коли Java-розробник намагається побудувати GUI в Swing або розробник на Visual C++ намагається створити користувальницький інтерфейс в MFC. Розробники часто не мають необхідних навичок для створення зручних і привабливих інтерфейсів. Ця робота більше підходить для дизайнерів інтерфейсів. Хороші дизайнери інтерфейсів краще знають, чого бажають користувачі, ніж експерти в області проектування і написання коду. Зрозуміло, буде краще, якщо

дизайнер інтерфейсів створить інтерфейс, а розробник напише код, який реалізує логіку цього інтерфейсу, але технології типу Swing або MFC просто-напросто не дозволяють чинити таким чином [10].

Для реалізації продукту було вирішено обрати ряд сторонніх бібліотек, що дозволять спростити реалізацію деяких компонентів програмного забезпечення. Було обрано такі бібліотеки:

- SharpNL – портована на .NET Framework бібліотека Apache OpenNLP, що реалізує інструменти для обробки текстових даних на мові програмування Java. Дана бібліотека підтримує виконання ряду задач, що є ключовими при обробці текстових даних [11]. Основними можливостями є:

  1. Токенізація.
  2. Розбиття на речення.
  3. Тегування (визначення лінгвістичних одиниць).

З огляду на обрану технологію, для розроблення програмного продукту доцільно зупинитись на виборі мови програмування, на якій буде написаний програмний продукт.

Технологія WPF дозволяє розроблювати програмні продукти на таких мовах програмування:

- C#;
- VB.NET;
- Microsoft Visual C++;
- IronRuby;
- IronPython;
- F#.

Згідно з поставленими задачами для розробки доцільно використати мову програмування C#, що є об'єктно-орієнтованою мовою програмування.

CC # (вимовляється як "See Sharp") - це проста, сучасна, об'єктно-орієнтована та безпечна для використання мова програмування. C # має

своє коріння в сімействі мов C і буде відразу знайомий програмістам C, C ++, Java та JavaScript.

C # - це об'єктно-орієнтована мова, але C # також містить підтримку компонентного програмування. Сучасний дизайн програмного забезпечення все частіше залежить від програмних компонентів у вигляді самодостатніх і самореалізаційних пакетів функціональності. Ключ до таких компонентів полягає в тому, що вони представляють модель програмування з властивостями, методами та подій; вони мають атрибути, що надають декларативну інформацію про компонент; і вони включають власну документацію. C # надає мовні конструкції, які безпосередньо підтримують ці поняття, що робить C # дуже природною мовою, в якій можна створювати та використовувати компоненти програмного забезпечення.

Деякі C# мають допомогу в побудові надійних і довговічних додатків: збирання сміття автоматично повертає пам'ять, зайнята недоступними невикористаними об'єктами; обробка винятків забезпечує структурований і розширюваний підхід до виявлення та відновлення помилок; і типовий дизайн мови робить неможливим читати з неініціалізованих змінних, індексувати масиви за межі їхньої межі або виконувати неперевірені типові листи.

C# має єдину систему типів. Всі типи C #, включаючи примітивні типи, такі як int та double, успадковуються від одного типу кореневого об'єкта. Таким чином, всі типи поділяють набір загальних операцій, а значення будь-якого типу можуть зберігатися, транспортуватися та працювати на постійній основі. Крім того, C# підтримує як призначені користувачем типи посилань, так і типи значень, що дозволяє динамічно розподіляти об'єкти.

Microsoft Visual Studio – серія продуктів фірми Майкрософт, які включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Ці продукти дозволяють розробляти як консольні програми, так і програми з графічним інтерфейсом, в тому числі з підтримкою технології Windows Forms, а також веб-сайти, веб-застосунки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight.

Visual Studio 2015 – найновіша версія Microsoft Visual Studio, під кодовим ім'ям Dev14 була представлена 20 червня 2015 року. Суттєвою зміною стала підтримка багатьох цільових платформ: окрім базової Windows з'явилась можливість будувати проекти для IOS та Android. Для розробників комп'ютерних ігор була додана підтримка фреймворку Unity. Був оновлений механізм автентифікації: користувач під час запуску Visual Studio синхронізується з єдиним аккаунтом Microsoft.

Версія включає в собі .NET Framework 4.6 та підтримку універсальної платформи Windows 10. Розробників на мові C++ потішили новими функціональними можливостями стандарту C++14, та навіть деякими поліпшеннями з C++17.

Останнім оновленням на данний момент є Update 2 від 30 березня 2016-го року, у якому багато уваги приділено стабільності, та продовжено роботу у напрямку підтримки нових стандартів мови C++.

Отже, за результатами огляду технологій для розробки було обрано платформу .NET Framework та мову програмування C#. Дані технології дозволять створити ефективні Windows-застосунки.

Для реалізації інтерфейсу користувача обрано бібліотеку Windows Presentation Foundation, що надає потужні можливості для створення складних інтерфейсних рішень.

В процесі розробки архітектури застосунку було вирішено використати ряд сторонніх бібліотек, що реалізують необхідні для реалізації системи можливості.

В якості середовища розробки обрано Microsoft Visual Studio, що надає інструменти для розробки на мові програмування C#, візуальне середовище розробки інтерфейсів користувача для бібліотеки WPF.

3.2 Архітектура розробленого програмного забезпеченя

З огляду на область застосування автоматизованої системи, що розроблюється було вирішено, що архітектура системи повинна бути побудована за клієнт-серверною моделлю. При цьому передбачається, що користувач встановлює на комп'ютері клієнт-застосунок, що проводить тільки мінімальну попередню обробку документа та підготовлює його для відправлення на сервер. Сервер в свою чергу проводить аналіз документів та відправляє на клієнстьку частину результуючі метадані з результатами роботи реалізованого методу. Клієнтська частина проводить рендеринг цих результатів на вхідних документах та відображує дану інформацію в зручному для користувача вигляді.

Клієнт-серверну архітектуру можна означити, як концепцію інформаційної системи в якій основна частина її ресурсів зосереджена в серверах, обслуговуючих клієнтів. Така архітектура визначає такі типи компонентів:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

Правила взаємодії між клієнтом і сервером називаються протоколом заємодії.

Модель клієнт-серверної взаємодії визначається перш за все розподілом обов’язків між клієнтом та сервером. Логічно можна відокремити три рівні операцій:

- рівень представлення даних, який по суті являє собою інтерфейс користувача і відповідає за представлення даних користувачеві і отрмання від нього керуючих команд та даних;
- прикладний рівень, який реалізує основну логіку застосунку і на якому здійснюється необхідна обробка інформації;
- рівень доступу до даних, який забезпечує зберігання даних та доступ до них.

Модель клієнт-сервер не вказує, що сервер-хости повинні мати більше ресурсів, ніж клієнт-хост. Скоріше за все, це дозволяє будь-якому комп'ютеру загального призначення розширити свої можливості, використовуючи спільні ресурси інших хостів. Централізовані обчислення, однак, спеціально виділяють велику кількість ресурсів на невелику кількість комп'ютерів. Чим більше обчислень вивантажується з клієнтських хостів на центральні комп'ютери, тим простіше може бути клієнт-хост. Він сильно залежить від мережевих ресурсів (серверів та інфраструктури) для обчислення та зберігання. Бездисковий вузол завантажує навіть свою операційну систему з мережі, а комп'ютерний термінал взагалі не має операційної системи; це лише інтерфейс вводу / виводу для сервера. Навпаки, жирний клієнт, такий як персональний комп'ютер, має багато ресурсів і не залежить від сервера для виконання важливих функцій.

Оскільки мікрокомп'ютери знижували ціну та збільшували потужність з 1980-х до кінця 1990-х років, багато організацій перекладали обчислення з централізованих серверів, таких як мікропроцесори та міні-комп'ютери, до жирних клієнтів. Це забезпечило більший, індивідуалізований панування над ресурсами комп'ютера, але ускладнило управління інформаційними технологіями. Протягом 2000-х років веб-програми витримали достатньо, щоб конкурувати з прикладними програмами, розробленими для конкретної мікроархітектури. Це дозрівання, більш доступне зберігання великих мас, а також поява сервіс-орієнтованої архітектури були одним з факторів, які породжували тенденцію хмарних обчислень у 2010-х роках.

3.3 Особливості програмної реалізації застосунку

Так як основна ідея модифікації базового методу полягає в додаванні кроків, які б виконували видалення входження синонімів із вхідних документів, основною задачею при реалізації даного модифікованого методу стало якомога більше пришвидшення кроку, що додається, для того, щоб вплив на швидкодію модифікованого методу в порівнянні з базовим методом був якнайменшим.

Аналізуючи запропоновану модифікацію, можна зробити висновок, що найбільший вплив на швидкодію модифікованого методу буде мати пошук в словнику синонімів. Тому при реалізації найбільший акцент потрібно зробити на максимальне пришвидшення пошуку в словнику синонімів. Цього можна добитися кількома шляхами:

- Застосування спеціалізованих структур даних, що спроектовані спеціально для виконання операцій пошуку на великих об’ємах даних
- Підготовка, або попередня обробка словника синонімів, структуризація, що пришвидшить пошук.

Розглянемо кожен з цих етапів більш детально. До спеціалізованих структур даних та алгоритмів що підтримують пошук на великих об’ємах даних можна віднести хеш-таблиці, бінарні дерева пошуку, кучі.

Геш-таблиця — структура даних, що реалізує інтерфейс асоціативного масиву, а саме, вона дозволяє зберігати пари (ключ, значення) і здійснювати три операції: операцію додавання нової пари, операцію пошуку і операцію видалення за ключем. снує два основних варіанта геш-таблиць: з ланцюжками і з відкритою адресацією. Геш-таблиця містить в собі деякий масив H, елементами якого є пари (геш-таблиця з відкритою адресацією) або списки пар (геш-таблиця з ланцюжками).

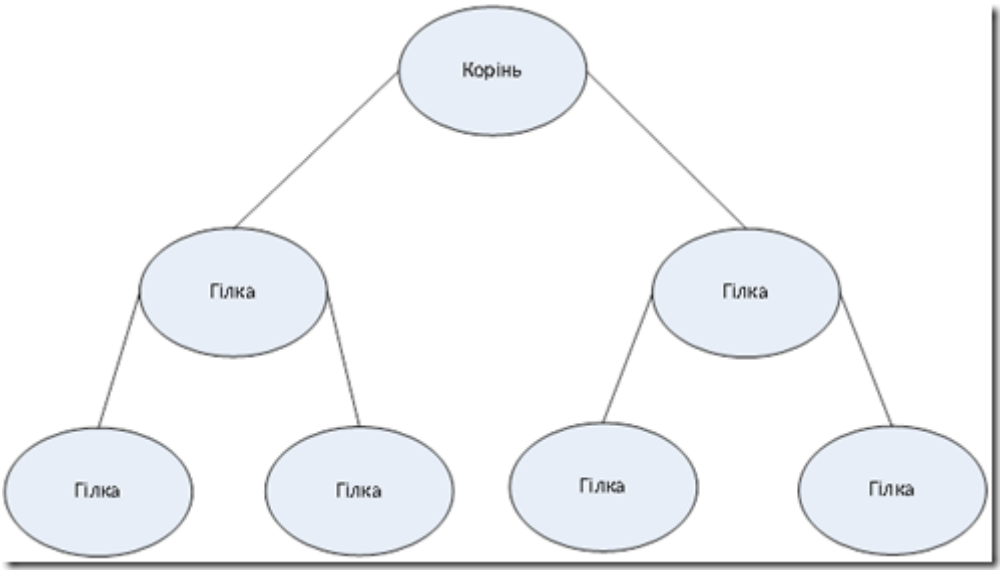
Виконання операцій в геш-таблиці починається з обчислення геш-функції від ключа. Отримане геш-значення i = hash(key) відіграє роль індексу в масиві H. Після цього операція (додавання, видалення, пошук) перенаправляється об'єктові, який зберігається у відповідній комірці масиву H[i].

Ситуація, коли для різних ключів отримується одне й те саме геш-значення, називається колізією. Такі події непоодинокі — наприклад, при додаванні в геш-таблицю розміром 365 комірок усього лише 23-х елементів ймовірність колізії вже перевищує 50 відсотків (якщо кожний елемент може з однаковою ймовірністю потрапити в будь-яку комірку) — див. парадокс днів народження. Через це механізм розв'язання колізій — важлива складова будь-якої геш-таблиці.

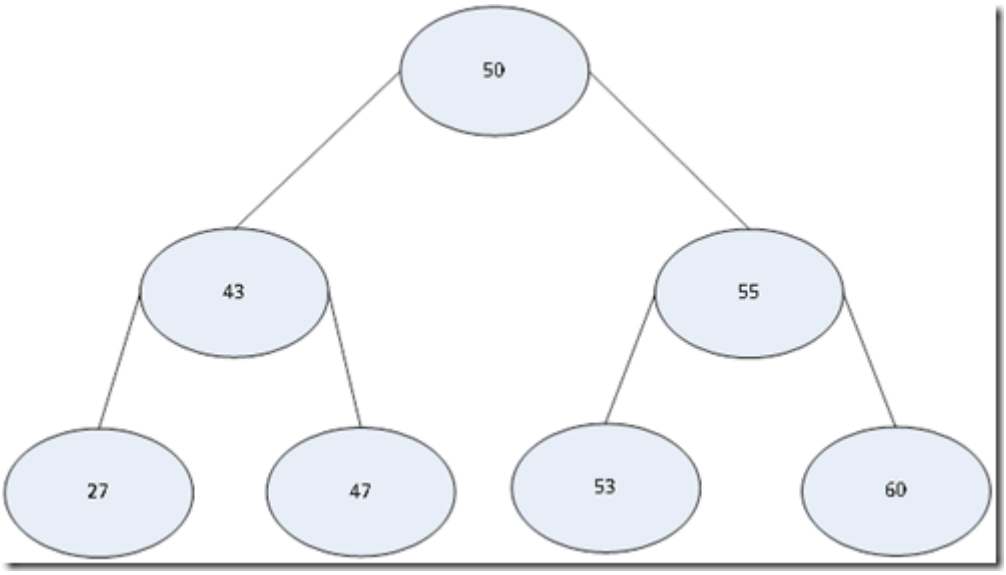
В деяких особливих випадках вдається взагалі уникнути колізій. Наприклад, якщо всі ключі елементів відомі заздалегідь (або дуже рідко змінюються), тоді для них можна знайти деяку досконалу геш-функцію[en], яка розподілить їх за комірками геш-таблиці без колізій. Геш-

таблиці, які використовують подібні геш-функції, не потребують механізму розв'язання колізій, і називаються геш-таблицями з прямою адресацією.

Бінарне дерево пошуку – це структура даних, яка призначена для того, щоб реалізувати швидкий пошук елементів, які зберігаються в цій структурі. Слово “бінарне” тут використовується для того, щоб вказати, що кожна гілка дерева може містить тільки дві дочірніх гілки. Структура бінарного дерева виглядає наступним чином:



Тепер, для того щоб зрозуміти різницю між бінарним деревом і бінарним деревом пошуку розгляньте наступну діаграму:



На попередній діаграмі зображено те саме дерево, де кожна гілка містить якийсь ключ (в нашому випадку ціле число). Зверніть увагу на те, що це дерево спроектовано таким чином, що кожна гілка містить дочірні гілки з більшим або з меншим значенням ключа. Наприклад гілка, яка містить ключ 43, містить дві дочірні гілки з меншим значенням ключа – 27 (ліва гілка) та більшим значенням ключа – 47 (права гілка).

Куча - це спеціалізована структура даних типу дерево, яка задовольняє властивості купи: якщо B є вузлом-нащадком вузла A, то ключ (A) ≥ ключ (B). З цього випливає, що елемент з найбільшим ключем завжди є кореневим вузлом купи, тому іноді такі купи називають max-купами (в якості альтернативи, якщо порівняння перевернути, то найменший елемент буде завжди кореневим вузлом, такі купи називають min-купами). Не існує ніяких обмежень щодо того, скільки вузлів-нащадків має кожен вузол купи, хоча на практиці їх число зазвичай не більше двох. Купа є максимально ефективною реалізацією абстрактного типу даних, який називається чергою з пріоритетом. Купи мають вирішальне значення в деяких ефективних алгоритмах на графах, таких, як алгоритм Дейкстри на d-купах і сортування методом піраміди.

РОЗДІЛ 4 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

4.1 Методика оцінювання ефективності методів визначення нечітких дублікатів

Для оцінки якості роботи методів визначення нечітких дублікатів застосовуються різноманітні метрики, що грунтуються на аналізі результатів роботи методу. При цьому ідеальними результатами вважаються такі, що повністю співпадають з думкою експертів, що також проводять оцінку. Існують такі метрики оцінювання якості:

— повнота (recall);

— точність (precision);

— акуратність (accuracy);

— F-міра.

Дані метрики базуються на матриці класифікації, наданій в таблиці 4.1

Таблиця 4.1 — Матриця класифікації

	Є дублікатом	Не є дублікатом
Знайдено методом	*a*	*b*
Не знайдено методом	*c*	*d*

Де *a* – кількість документів, знайдених методом і є дублікатами з точки зору експертів; *b* – кількість документів, знайдених методом, але яні не є дублікатами на думку експертів; *c* – кількість документів, що є дублікатами з точки зору експертів, але не знайдених методом; *d* – документи, що не є дублікатами з точки зору експертів і виявлені методом як не дублікати.

Повнота

Повнота (recall) обчислюється як відношення знайдених дублікатів до їх загальної кількості:

$$R = \frac{a}{a + c}$$

Повнота характеризує можливість методу знаходити дублікати, але не враховує кількість документів, які не є дублікатами з точки зору експертів. Наприклад, якщо повнота дорівнює 75% то це означає, що 25% дублікатів не знайдено.

Точність

Точність (precision) визначається як відношення знайдених дублікатів до загальної кількості документів:

$$P = \frac{a}{a + b}$$

Точність характеризує можливості методу видавати в списку результатів тільки дублікати. Наприклад, якщо точність дорівнює 25% то це означає що із усього набору результатів тільки чверть окументів є дублікатами.

Акуратність

Акуратність (accuracy) визначається відношенням правильно знайдених методом дублікатів до загального числа документів:

$$A = \frac{a + d}{a + b + c + d}$$

F-міра

F-міра використовується для того, щоб об'єднати показники точності і повноти в одній величині. Для цього F-міра обчислюється за формулою:

$$F_\beta = \frac{(1 + \beta^2) * P * R}{\beta^2 * P + R}$$

При $\beta = 1$ F-міра надає однакову вагу як точності так і повноті і називається збалансованою, або F_1 -мірою. Для неї формула значно спрощується:

При дослідженнях дана метрика використовується як єдина метрика, по значенню якої можна робити висновок про якість роботи методу.

4.2 Вибір наборів даних для тестування розробленого методу

Для дослідження ефективності розробленого модифікованого методу визначення нечітких дублікатів потрібно обрати набори даних, які добре підходять для такого роду аналізу.

WT10G

Для дослідження дублікатів у веб-документах було вирішено обрати корпус WT10G. Він створювався університетом Глазго протягом кількох років шляхом зібрання веб-сторінок з 11680 серверів. При цьому було зібрано корпус, що включає 1692096 документів. Однією з найбільших переваг цього корпусу є проведений аналіз щодо зв'язків між документами. Додатково на базі даного корпусу було проведене дослідження з близькості документів за змістом, що безперечно є перевагою в аналізі ефективності розробленого методу так як дані оцінки можна вважати експертними оцінками в обраних метриках для дослідження.

Корпус доступний у вільному доступі на веб-сайті університету Глазго та включає в собі 1,692,096 документів поділених на 104 директорії. Кожен файл включає в себе документи у спеціальному форматі SGML, що відомий як TREC веб-формат. TREC — серія конференцій, що займається питаннями інформаційного пошуку та інших задач. Зокрема, для цієї серії конференцій було розроблено зазначений формат вхідних документів. Формат передбачає, що кожен документ є підтипом sgml, тому

є схожим на XML або HTML, передбачає те, що кожен документ має починатись спеціальним відкриваючим тегом і закінчуватись закриваючим тегом. При цьому існують додатковий набір тегів, що включає TEXT, HL, HEAD, HEADLINE, TTL, та LP для форматування самого документу і розділення його на окремі логічні частини.

Corporate Messaging dataset

CMD – частина Common Crawl dataset, що збирався протягом 7 років і включає більше петабайту даних і включає в себе веб-сторінки, метадані та виділений текст даних веб-сторінок.

Corporate Messaging розміщений на популярному сховищі Amazon S3 та доступний для вільного завантаження. Набір даних Common Crawl живе на Amazon S3 як частина програми Amazon Public Datasets.

З Public Data Sets ви можете завантажувати ці файли абсолютно безкоштовно за допомогою HTTP або S3.

Оскільки "Common Crawl Foundation" розвивається протягом багатьох років, він також має формат та метадані, які супроводжують сканування.

Common Crawl в даний момент зберігає дані сканування за допомогою веб-ARChive (WARC) формату.

До цього моменту сканування зберігалося у форматі файлу ARC.

Формат WARC дозволяє більш ефективно зберігати та обробляти безкоштовні веб-архіви з декількох мільярдів веб-сторінок Common Crawl, які можуть бути розміром до сотні терабайт.

- WARC-файли, які зберігають необроблені дані сканування
- WAT-файли, в яких зберігаються обчислювальні метадані для даних, що зберігаються в WARC
- WET-файли, які зберігають витягнутий відкритий текст із даних, що зберігаються в WARC

Формат WARC - це вихідні дані з сканування, що забезпечує безпосереднє відображення процесу сканування. Формат не лише зберігає HTTP-відповідь на веб-сайтах, на яких він контактує (WARC-Type: response), але також зберігає інформацію про те, як ця інформація запитується (WARC-Type: request) та метадані в самому процесі сканування (WARC-Type : метадані).

4.3 Результати роботи модифікованого методу

Основними параметрами досліджень результатів роботи методу були швидкість роботи на однакових корпусах даних та набір метрик, що визначають ефективність роботи методу. В якості даних метрик було обрано метрики, що застосовуються для оцінювання методів класифікації та засновані на визначенні false-positive та false-negative значень в порівнянні з експертною оцінкою.

Тестування проводилось для:

- базового I-Match методу
- модифікованого I-Match методу, запропонованого в даній дисертації. Також, виходячи із специфіки запропонованої модифікації було застосовано відповідний словник синонімів, що був додатково підготований і переформатований для роботи з запропонованим методом.
- Методу шинглів в реалізації бібілотеки text-shingles, що доступна на github
- методу winnowing в реалізації однойменної бібліотеки winnowing

Для тестування розробленого модифікованого методу пошуку нечітких дублікатів використовувалась система наступної конфігурації:

- CPU: Intel Core i7-3612QM
- RAM: 6 Gb
- OS: Linux Debian 9 x64

Наступна таблиця показує швидкодію розробленого методу у порівнянні з існуючими аналогами:

Корпус	I-Match*	**I-Match (мод)***	Метод шинглів	Winnowing
WT10G	19.150	**21.117**	56.183	102.732
CMD	8.430	**9.012**	18.023	18.340
Crosswikis	7.236	**8.120**	17.620	12.874

Табл.4.2 швидкодія протестованих методів

З результатів роботи ми можемо бачити, що модифікований метод втрачає у швидкодії у порівнянні з своїм базовим аналогом. Це обумовлено додаванням кроку з пошуку синонімів при побудові “відбитку” документа. При цьому ми бачимо, що втрати в швидкодії є не досить суттєвими у порівнянні з іншими аналогами. В середньому спостерігається втрата в швидкодії близько 10%. Враховуючи той факт, що більшість методів не ставлять перед собою за мету покращення швидкодії та зазвичай користувачі даних методів не потребують надшвидких результатів, то дані втрати в швидкодії є не досить суттєвими.

В наступній таблиці проведено аналіз точності на корпусі WT10G із застосуванням описаних раніше метрик:

Метод	Повнота	Точність	Акуратність	$F_{0.5}$-міра
I-Match	0,894	0,927	0,912	0,9
I-Match (мод).	**0,924**	**0,935**	**0,93**	**0,926**
Метод шинглів	0,824	0,803	0,811	0,819
Winnowing	0,75	0,733	0,739	0,746

Табл. 4.3 Значення метрик для корпусу WT10G

Для корпусу CMD отримано такі результати:

Метод	Повнота	Точність	Акуратність	$F_{0.5}$-міра
I-Match	0,824	0,915	0,874	0,84
I-Match (мод).	**0,814**	**0,927**	**0,875**	**0,834**
Метод шинглів	0,864	0,822	0,839	0,855
Winnowing	0,75	0,733	0,739	0,746

Табл. 4.4 Значення метрик для корпусу CMD

Для корпусу Crosswikis отримано такі результати:

Метод	Повнота	Точність	Акуратність	$F_{0.5}$-міра
I-Match	0,768	0,739	0,749	0,762
I-Match (мод).	**0,734**	**0,731**	**0,732**	**0,733**
Метод шинглів	0,758	0,735	0,743	0,753
Winnowing	0,768	0,738	0,748	0,761

Табл. 4.5 Значення метрик для корпусу Crosswikis

Як можемо бачити з результатів модифікований метод має кращі показники метрик для більшості корпусів, на яких було протестовано методи. Для корпусу Crosswikis розроблений метод показав дещо гірші результати, це може бути зумовлено тим, що статті на ресурсі Wikipedia носять в більшості науковий характер та використання довгих синонімічних рядів там не характерне.

4.4 Висновки

У даному розділі проведено аналіз розробленого методу визначення нечітких дублікатів в тектсових даних. Метод дозволяє розв’язати задачу дослідження, даючи змогу створити автоматизовані системи пошуку нечітких дублікатів в природномовних текстових даних.

Проведено аналіз параметрів якості методів визначення образливого вмісту, що в даному випадку зводиться до бінарної класифікації. Виділено 4 основні метрики: частка правильно класифікованих текстів, точність, повнота та F-міра. Крім того було заміряно швидкодію алгоритму на визначеній конфігурації комп’ютера для оцінки втрат швидкодії при додаванні до базового методу відповідної модифікації.

Крім того, для тестування було відібрано та розмічено вибірку, що складається з 500 повідомлень з обраних корпусів, в яких було перемішано вміст для емулювання плагіату та 500 незмінених документів, в якості контрольної групи.

Завдяки використанню створеного програмного забезпечення було проведено аналіз показників якості визначення нечітких дублікатів запропонованим методом. Запропонований метод показав кращі на 10-15% результати точності та при проведенні ситнетичного тесту з заміною слів на синоніми у документах — на 50% кращі результати. При цьому втрати у швидкодії склали в середньому 10%, що є доволі гарним результатом, так як для методів визначення нечітких дублікатів швидкодія не є центральним показником ефективності.

РОЗДІЛ 5 ПОБУДОВА БІЗНЕС МОДЕЛІ

5.1. Опис проблеми

Розвиток Інтернету та активне впровадження цифрових технологій у повсякденне життя людини призвели до стрімкого збільшення цифрових текстових даних.

Виявлення нечітких дублікатів документів є однією з важливих і важких задач у галузі автоматизованого аналізу природномовних текстових даних. Її актуальність визначається різноманітністю застосувань, у яких потрібно враховувати «подібність» документів, наприклад — це й поліпшення якості пошукового індексу та архівів пошукових систем, об'єднання статей за подібністю їх змісту, фільтрація поштового й пошукового спаму, визначення порушень авторських прав тощо.

Отже, розроблений у даній роботі метод визначення нечітких дублікатів може бути застосований у багатьох сферах.

Визначення порушення авторських прав стала однією з перших сфер, де застосовувались та для якої розроблялись нові методи визначення дублікатів. Саме тому першою асоціацією, коли ми чуємо про методи визначення дублікатів, є боротьба за авторські права. Але не завжди дані продукти повинні використовуватись як засоби пост-аналізу вже написаних робіт як захист від неавторизованого копіювання. Також дані системи можуть надати науковцям шлях розширення кола джерел, з яких можна дістати інформацію для роботи в процесі її написання.

На сьогоднішній день пошукові індекси стали одними з найважливіших систем у всій мережі Інтернет. При цьому обсяги даних, що входять до цих індексів перевищують усі можливі об’єми, досяжні для обробки вручну. При цьому існує проблема дублікації входжень у пошуковий індекс для документів, що мають дуже багато спільного. Застосування методів визначення нечітких дублікатів для цієї сфери є

досить новою ідеєю, їх застосування дозволить зменшити обсяги інформації, що входять в пошукові індекси та пришвидшити і покращити пошук в ньому.

Всі описані вище проблеми узагальнені в дереві проблем, що зображено на рис. 5.1.

5.2. Зацікавлені сторони

У вирішенні зазначений вище проблем зацікавлено певне коло осіб.

Перш за все це різноманітні пошукові системи. Це можуть бути як системи загального призначення, що працюють з різноманітними видами текстових даних, так і спеціалізовані системи, що працюють з одноманітними даними і надають послуги з пошуку. Не дивлячись на те, що дані представники можуть здатись подібними на перший погляд, вони прагнуть вирішення двох різних проблем: пошукові системи загального призначення у більшості випадків мають величезні обсяги інформації і перш за все прагнуть скоротити ці об’єми при збереженні поточних показників точності пошуку; в свою чергу – спеціалізовані пошукові системи не мають таких великих об’ємів даних, приріст нової інформації в них може бути зовсім невеликим. Вони перш за все прагнуть збільшити точність пошуку на існуючих наборах даних.

Також в вирішенні проблеми зацікавлені сторони, що надають текстові документи до спеціалізованих пошукових систем. Висока швидкість дозволить їм оцінити документи на відповідність прийнятним підприємствами критеріям. У разі невідповідності, вони зможуть переробити текстовий документ та зробити повторну оцінку, не витрачаючи час на очікування результату.

Опосередковано в вирішенні описаної проблеми зацікавлені такі сторони, як держава, замовники і інші. Держава зацікавлена в покращенні роботи певних спеціалізованих систем (зокрема пошукових систем в юридичній сфері)

Замовники зацікавлені у більш швидкому виконанні їх замовлень. Проте ці зацікавлені сторони мають невеликий вплив.

Матриця зацікавлених осіб

Група зацікавлених осіб	Інтереси зацікавленої особи	Вплив	Стратегії приваблення
Пошукові системи заг. призначення	Скорочення обсягів збереженої інформації	Великий	Рекламні матеріали. Презентації продукту, демонстрації. Участь в тематичних заходах. Надання безкоштовних пробних версій. Забезпечення технічної підтримки.
Спеціалізовані пошукові системи	Підвищення ефективності пошуку на наявних об’ємах даних.	Великий	
Сторони, що надають матеріали підприємствам	Зменшення часу отримання результатів при попередній оцінці документів на відповідність критеріям	Середній	
Держава	Збільшення ефективності спеціалізованих пошукових систем в держ. власності	Середній	
Замовники	Збільшення швидкості виконання замовлень	Малий	

5.3. Комерційне рішення. Основні характеристики

Поставлену проблему може вирішити спеціалізований програмний продукт, що реалізує описану технологію пошуку нечітких дублікатів в текстових документах. Завдяки описаному в даній роботі алгоритму, текстові дані оброблюються та зберігаються ефективно, що дозволяє збільшити швидкість пошуку, забезпечуючи при цьому прийнятну точність.

Відповідно, ці параметри підвищать швидкість пошуку плагіату в текстових документах, що є важливим для кінцевих користувачів.

Основними клієнтами є університети, видавництва, стрічки новин та paper mill – організації, що спеціалізуються на написанні дипломних, курсових та інших робіт для студентів. Саме тому програмне рішення повинно бути представлене у вигляді веб-сервісу, що буде мати зовнішні API. На їх базі буде побудований веб-сайт, що буде задовольняти потребам більшої кількості користувачів. Завдяки API забезпечується гнучкість рішення, наприклад, великі організації зможуть легко інтегрувати сервіс у свої існуючі програмні комплекси. Внутрішня архітектура продукту розроблена з оглядом на легку масштабованість, здатність змінювати потужності в залежності від кількості активних користувачів автоматично, що дозволить зменшити витрати на підтримання постійних комп'ютерних потужностей та персоналу для обслуговування.

5.4. Конкурентні переваги рішення

Конкурентів даного програмного рішення на ринку досить багато. Умовно їх можливо розділити на дві великі групи:

- веб-сервіси у вигляді сайтів;
- спеціалізовані програмні комплекси.

Веб-сервіси у вигляді сайтів, частіше всього, розраховані на індивідуальне використання та не підходять для використання корпоративним клієнтам та клієнтам, що часто змінюють текстові документи великого об’єму та роблять періодичні перевірки. Також, зазвичай, такі сервіси не мають великої бази даних документів для звірки, тому результати даних продуктів можуть бути досить неточними якщо досліджуваний документ не потрапляє за тематикою до наявного в сервісу банку документів. Це доволі великий недолік даних сервісів і користувачі змішені перевіряти документи на кількох сервісах, що спеціалізуються на різних тематиках, аби бути впевненими, що їх документ не містить запозичень.

Спеціалізовані програмні комплекси є досить залежними від обчислювальних потужностей, наявних даних у організації або відкритих баз даних та використовуються, зазвичай, корпоративними клієнтами. Недоліками є пряма залежність вид наявності потужностей та кількості вхідних даних. Часто програмні комплекси мають велику базу даних текстових документів, що, безперечно, є перевагою, але через складність реалізації та інтеграції в копроративну систему організації швидкість роботи з нею є досить малою, коли користувачу може знадобитись цілий день аби перевірити документ, що, очевидно, є дуже вагомим недоліком.

Отже, конкурентними перевагами програмного продукту, що пропонується є:

- висока швидкість опрацювання текстових документів за рахунок запропонованого модифікованого методу;
- ефективне збереження текстових документів та можливість створення як глобальної бази так і спеціалізованих корпусів;
- висока точність в порівнянні з методами, що мають схожу швидкодію;
- можливість легкого інтегрування сервісу у існуючі програмні комплекси;
- можливість вдосконалення сервісу під конкретного замовника.

5.1. Клієнти. Сегменти ринку споживання

Як вже зазначалось вище, потенційних клієнтів продукту можна поділити на два сегменти – корпоративні та індивідуальні клієнти.

Відповідно, до кожного з цих сегментів пропонується індивідуальний підхід, що базується на потребах сегменту.

Сегмент корпоративних клієнтів характеризується необхідністю частої обробки великих об’ємів документів та частою наявністю власних наборів документів для порівняння. Для даного сегменту неприйнятна окрема плата за кожен документ, що оброблюється, тому для даного сегменту доцільним є запропонування підписки, що базується на часовому проміжку. В такому випадку організація може не хвилюватись за кількість оброблюваних документів. При цьому дані клієнти будуть доволі стабільним потоком дохідів та зможуть надати багато зворотньої інформації з можливостей вдосконалення та підготування продукту під конкретного корпроративного користувача.

Для сегменту індивідуальних клієнтів ціннісною пропозицією є наявність декількох варіантів підписок для використання системи, що дозволяє більш гнучко підлаштовувати використання сервісу під конкретні

потреби клієнта, також, використовуючи сервіс, клієнт отримує можливість аналізу на великій базі, що динамічно буде наповнюватись. Це може бути як часова підписка для індивідуальних користувачів, що потребують обробки великих об’ємів даних, так і форма оплати за один оброблюваний документ.

5.2. Доходи та витрати

Сумарний дохід складається як сума доходів від продаж товарів та супутніх послуг для кожного сегменту споживачів.

Для сегменту корпоративних клієнтів пропонується продавати наступні товари та надавати наступні послуги:

- довготривалі підписки з динамічним налаштуванням максимальної кількості документів, що одночасно знаходяться в черзі;
- індивідуальне налаштування системи під конкретного клієнта;
- розширена технічна підтримка.

Під довготривалими підписками з динамічним налаштуванням максимальної кількості документів, що одночасно знаходяться в черзі мається на увазі, що корпоративний клієнт може обрати часовий проміжок підписки, можливість інтеграції продукту в корпоративну систему, цілодобову технічну підтримку.

Індивідуальне налаштування системи під конкретного клієнта є окремою послугою, що оплачується за кожен прецедент надання її та включає в себе корекцію параметрів алгоритму, використання або виключення специфічних баз, зміна інтерфейсу системи у відповідності до вимог замовника.

Розширення технічна підтримка пропонується як послуга на термін дії підписки і включає в себе гарантоване оброблення запитів в швидкі терміни в робочі та вихідні дні.

Для сегменту індивідуальних клієнтів пропонується продавати наступні товари:

- підписку з обмеженням на кількість документів для аналізу;
- підписку з короткотривалим часовим обмеженням та обмеження на один документ, що може оброблятись одночасно в будь-який момент часу.

Різниця між короткотривалими та довготривалими часовими обмеженнями полягає в тому, що короткотривалі включать в себе терміни до місяця такі як день, 3 дні, тиждень, 2 тижні, місяць, а довготривалі, відповідно, починаються від місяця – місяць, 3 місяці, півроку, рік т.д. Дані обмеження дозволять залучити до користування системою як довготривалих користувачів, для яких потрібне постійне користування системою, так і для короткочасних користувачів, таких як студенти, для яких перевірка на плагіат може знадобитись тільки в окремі періоди року.

Розрахований дохід по місяцям протягом першого року наведено в табл. 5.2. Всі суми наведено в доларах США.

Таблиця 5.2 Доходи

	Довготривалі підписки	Індивідуальне налаштування	Розширена технічна підтримка	Підписка з обмеженням на кількість документів	Короткотривалі підписки	Всього
1	180	0	0	50	90	320
2	420	300	100	200	300	1320
3	800	500	400	400	450	2550
4	1100	800	500	450	600	3450
5	1500	1200	600	500	1000	4700
6	2400	1800	1750	750	1500	8200
7	2000	1600	800	700	1400	5500
8	1900	1700	700	600	1200	6100
9	3000	2200	1100	1100	1500	8900
10	1900	1800	700	700	1100	6200
11	2000	2200	900	900	1000	6800
12	2600	2400	1000	1300	1100	7200
Сумарно за рік:						61240

Сукупність загальних витрат складають наступні витрати:

- витрати на розроблення, підтримку та вдосконалення програмних продуктів (утримання робочих місць, придбання необхідних інструментів та програмних засобів);
- витрати на надання послуг технічної підтримки (утримання робочих місць, придбання необхідних інструментів та програмних засобів);
- витрати на оплату праці (заробітна плата та інші виплати працівникам);
- витрати на оренду офісних приміщень;
- витрати на опалення, освітлення, водопостачання, водовідвід;
- адміністративні витрати (витрати на зв'язок, податки та збори, плата за послуги банків тощо);
- витрати на рекламу та дослідження ринку.

На початкове розроблення необхідно 3000$, також доробка буде відбуватись протягом 11 місяців і потребує суму у розмірі 1650$. Отже, сумарно на розробку буде витрачено 4650$.

Витрати по місяцям протягом першого року наведено в табл. 5.3. Всі суми наведено в доларах США.

Таблиця 5.3

Витрати

	Технічна підтримка	Оплата праці	Комунальні та адміністративні	Реклама	Всього
1	500	2000	500	350	3350
2	50	2000	500	400	2950
3	50	2000	500	450	3000
4	50	2000	500	500	3050
5	50	2000	500	700	3250
6	50	2000	500	700	3250
7	50	2000	500	700	3250

8	50	2000	500	700	3250
9	50	2000	500	700	3250
10	50	2000	500	700	3250
11	50	2000	500	700	3250
12	50	2000	500	700	3250
Сумарно за рік:					38350

Розрахуємо маржинальний прибуток за перший рік за формулою (5.1) [5].

$$\text{Маржинальнийприбуток} = \text{Дохід} - \text{Витрати} \quad (5.1)$$

За формулою (5.1) маржинальний прибуток за перший рік складе 61240-(4650+38350) = 18240 доларів США.

5.3. Бізнес-модель

Узагальнимо вище наведену інформацію в виглядів блоків, що складають так звану канву бізнес-моделі або lean canvas [6].

Споживачі:

- ранні клієнти: малі підприємства та окремі індивідуальні клієнти;
- середні і великі підприємства, індивідуальні клієнти.

Проблема:

- низька швидкість аналізу текстових документів на плагіат;
- велика обчислювальна складність алгоритмів;
- значне використання ресурсів;
- не завжди точний пошук.

Рішення:

- програмний продукт, що має високу швидкість аналізу та зберігання текстових документів при гарній точності.

Унікальна ціннісна пропозиція:

– гнучкі підписки для корпоративних і індивідуальних клієнтів;
– можливість легкої інтеграції в існуючі системи і індивідуального налаштування.

Потоки доходів:

– продаж довготривалих підписок для корпоративних клієнтів з динамічним налаштуванням максимальної кількості документів, що одночасно знаходяться в черзі;
– продаж послуг індивідуального налаштування системи під конкретного клієнта;
– продаж розширеної технічної підтримки;
– продаж підписки з обмеженням на кількість документів для аналізу для індивідуальних клієнтів;
– продаж підписки з короткотривалим часовим обмеженням та обмеження на один документ, що може оброблятись одночасно в будь-який момент часу для індивідуальних клієнтів.

Структура витрат:

– витрати на розроблення, підтримку та вдосконалення програмних продуктів;
– витрати на надання послуг технічної підтримки;
– витрати на оплату праці;
– витрати на опалення, освітлення, водопостачання, водовідвід;
– адміністративні витрати;
– оплата праці висококваліфікованих працівників;
– витрати на рекламу та дослідження ринку.

Також в канву бізнес-моделі включаються структурні блоки, які ще не були розглянуті. Це прихована перевага (перевага, яку не можливо скопіювати або купити), ключові метрики (основні показники, що вимірюються) та канали (шляхи до користувачів).

В якості каналів контакту з клієнтами пропонується використовувати наступні шляхи: тематичні ресурси та спільноти, конференції, профільні видання, прямі контакти для продажів.

Прихованою перевагою нашого продукту виступає використання технології, що базується на новому методу визначення нечітких дублікатів в текстових даних.

Ключовими метриками є наступні: кількість проданих підписок різних типів, кількість продажів індивідуальних налаштувань системи, кількість запитів в службу технічної підтримки, середній час обробки документа відносно його об'єму.

Описані структурні блоки об’єднано в єдину канву бізнес-моделі, котра наведена в табл. 5.4.

Таблиця 5.4

Канва бізнес-моделі

Проблема	**Рішення**	**Унікальна ціннісна пропозиція**	**Прихована перевага**	**Споживачі**
Низька швидкість аналізу текстових документів на плагіат. Велика обчислювальна складність алгоритмів.	Програмний продукт, що має високу швидкість аналізу текстових документів при гарній точності визначення дублікатів	Гнучкі підписки для корпоративних і індивідуальних клієнтів. Можливість легкої	Використання технології, що базується на новому алгоритмі для пошуку плагіату серед текстових документів	Ранні клієнти: малі підприємства та окремі індивідуальні клієнти. Середні і великі підприємства, індивідуальні клієнти.

<table>
<tr><td>Значне використання ресурсів.

Не завжди точний пошук.</td><td>Ключові метрики
Кількість проданих підписок різних типів.

Кількість продажів індивідуальних налаштувань системи
Кількість запитів в службу технічної підтримки..
середній час обробки документа відносно його об'єму</td><td>інтеграції в існуючі системи і індивідуального налаштування</td><td>Канали
Тематичні ресурси та спільноти.

Профільні видання.

Прямі контакти для продажів

конференції</td><td></td></tr>
<tr><td colspan="3">Структура витрат
Витрати на розроблення, підтримку та вдосконалення продуктів.
Витрати на надання послуг технічної підтримки.
Оплата праці розробникам та іншим працівникам.
Комунальні послуги.
Адміністративні витрати.
Витрати на рекламу та дослідження ринку.</td><td colspan="2">Потоки доходів
Продаж довготривалих підписок для корпоративних клієнтів з динамічним налаштуванням максимальної кількості документів, що одночасно знаходяться в черзі.
Продаж послуг індивідуального налаштування системи під конкретного клієнта.
Продаж розширеної технічної підтримки.
Продаж підписки з обмеженням на кількість документів для аналізу для індивідуальних клієнтів.</td></tr>
</table>

5.4. Висновки

В даному розділі розглянуто питання практичного застосування розробленого методу визначення нечітких дублікатів в природномовних текстових даних..

Незважаючи на те, що побудована бізнес-модель потребує доопрацювань, вона підтверджує життєздатність стартапу, заснованого на застосуванні алгоритму для пошуку нечітких дублікатів серед текстових

документів, що базується на модифікації методу I-Match, що в базовій конфігурації зарекоментував себе швидкістю тат точністю визначення при невеликих обчислювальних навантаженнях. Описані сегменти споживачів, які готові платити за вирішення даної проблеми, наявні унікальні ціннісні пропозиції для кожного сегменту. Запропонований програмний продукт має конкурентні переваги.

Розраховані доходи та витрати протягом першого року роботи. Знайдений маржинальний прибуток за цей період – 18.2 тисяч доларів США – також підтверджує життєздатність описаної бізнес-моделі.

ВИСНОВКИ

У даній роботі виконані наступні завдання:

1. Проведено аналіз існуючих методів визначення нечітких дублікатів в текстових даних та наявних комерційних програмних продуктів, що надають можливості з пошуку нечітких дублікатів в природномовних текстових даних.

2. Визначено основні характеристики, переваги та недоліки існуючих методів визначення нечітких дублікатів, на основі даних результатів визначено можливі шляхи модифікації методів.

3. Обрано базовий метод для модифікації, сформульовано можливі шляхи його модифікації.

4. Сформульовано модифікований метод визначення нечітких дублікатів, як такий, що підтримує синонімічні входження у вхідних документах.

5. Реалізовано програмне забезпечення для визначення нечітких дублікатів в текстових даних, що реалізує запропонований метод.

6. Проведено порівняльний аналіз ефективності сформульованого методу в порівнянні з існуючими аналогами та виконано оцінку отриманих результатів.

7. Сформульовано шляхи можливого подальшого вдосконалення запропонованого методу.

Результати, отримані при реалізації методу показали, що запропонований метод показує покращені результати якості в порівнянні з базовим методом, при цьому втрати швидкодії, спричинені додаванням додатковго кроку з видалення синонімічних входжень, не є значними.

Перспективний напрямком подальшого вдосконалення методу є дослідження можливості застосування різноманітних хеш-функцій через відсутність аргументації застосування хеш-функції SHA-1 (NIST 1995) в базовому методі.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Stanford NLP [Електронний ресурс] — Режим доступу: https://nlp.stanford.edu/IR-book/html/htmledition/tokenization-1.html (дата звернення 30.01.2018). — Назва з екрана.
2. Stanford NLP [Електронний ресурс] — Режим доступу: https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html (дата звернення 30.01.2018). — Назва з екрана.
3. TIOBE Index for May 2018 [Електронний ресурс] – Режим доступу: https://www.tiobe.com/tiobe-index/ – (09.10.2017).
4. Особенности языка C# [Електронний ресурс]. — Режим доступу : http://phpbb3.ru/?p=7831. — (09.02.2016).
5. Microsoft [Електронний ресурс]. — Режим доступу : https://www.microsoft.com/net/download/windows. — (09.02.2018).
6. .NET 2015 Overview [Електронний ресурс]. — Режим доступу : https://channel9.msdn.com/Events/Visual-Studio/Connect-event-2015/NET-2015-Overview. — (09.05.2018).
7. Платформа .Net та її застосування для ООП [Електронний ресурс]. — Режим доступу : http://www.znannya.org/?view=csharp-dotNET. — (09.02.2016).
8. Visual C++ Team Blog [Електронний ресурс]. — Режим доступу : https://blogs.msdn.microsoft.com/vcblog/2013/07/19/c99-library-support-in-visual-studio-2013/. — (09.05.2018).
9. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley. pp. 97ff. ISBN 0-201-63361-2.
10. Тлумачний словник з інформатики / Г.Г. Півняк, Б.С. Бусигін, М.М. Дівізінюк та ін. – Д., Нац. гірнич. ун-т, 2010. – С. 495][Міжнародний союз електрозв’язку; переклад – Юрій Пероганич (2006-11-23). Словник

термінів. Всесвітній саміт з питань інформаційного суспільства. Підсумкові документи. с. 84. Архів оригіналу за 2013-06-25

11. Семантичний аналіз тексту та пошук плагіату [Електронний ресурс] – дата візиту 26.11.2017. – Режим доступу до ресурсу: https://goo.gl/gJDW2C
12. Как работает метод шинглов при проверке текста на плагиат [Електронний ресурс] – дата візиту 28.11.2017. – Режим доступу до ресурсу:
13. Text-shingles [Електронний ресурс] – дата візиту 26.11.2017. – Режим доступу до ресурсу:
14. Бібліотека winnowing [Електрониий ресурс] – дата візиту 26.11.2017. – Режим доступу до ресурсу:
15. *Abdur Chowdhury* Collection Statistics for Fast Duplicate Document Detection / Abdur Chowdhury, Ophir Frieder, David Grossman, Mary C. McCabe // J Supercomput (2008) 45 — С. 255–276.
16. *A. Kołcz* Lexicon randomization for near-duplicate detection with I-Match / A. Kołcz, A. Chowdhury // ACM Transactions on Information Systems, Vol. 20, No. 2, April 2002 — С. 171–191.
17. *Jones K. S.* A statistical interpretation of term specificity and its application in retrieval / Jones K. S. // Journal of Documentation : журнал. — MCB University : MCB University Press, 2004. — Т. 60, № 5. — С. 493-502.
18. Геш-функція. Картка даних терміну : [арх. 23.09.2017] // Українське агентство зі стандартизації. — Дата звернення: 23.09.2017.

ДОДАТКИ

Додаток 1

Копія презентації

Додаток 2
Приклади програмного коду

```
using System;
using System.Collections.Generic;
using System.IO;
using System.IO.Compression;
using System.Linq;
using System.Reflection;
using System.Windows;
using Module = SummarizerSharp.Data.Core.Module;

namespace SummarizerSharp.Data
{
   public class ModuleLoader
   {
      private readonly string _directory;
      private readonly Dictionary<string, object> _globalParameters = new Dictionary<string,
object>();
      private readonly Dictionary<string, Module> _modules = new Dictionary<string, Module>();

      public ModuleLoader(string modulesDirectory)
      {
         _directory = modulesDirectory;
      }

      public IEnumerable<string> Names => _modules.Keys;

      public Module this[string name] => _modules[name];

      public IEnumerable<Type> ConfigTypes =>
         _modules.Values.Select(x => x.GetConfiguration()?.GetType()).Where(x => x != null);

      public void AddGlobalParameter(string name, object param)
      {
         _globalParameters.Add(name, param);
      }

      public void RemoveGlobalParameter(string name)
      {
         _globalParameters.Remove(name);
      }

      public void AddModule(string key, Module module)
      {
         module.Key = key;
         _modules.Add(key, module);
      }

      public object GetGlobalParameter(string name)
      {
         return _globalParameters[name];
      }

      private bool IsModuleExists(string name)
      {
         return Directory.Exists(Path.Combine(_directory, name)) &&
```

```
            File.Exists(Path.Combine(_directory, name, name + ".dll"));
    }

    private string GetModuleName(string name)
    {
        return Path.GetFullPath(Path.Combine(_directory, name, name + ".dll"));
    }

    private Module Load(string name, string path)
    {
        var DLL = Assembly.LoadFile(path);

        foreach (var type in DLL.GetExportedTypes())
        {
            if (type.IsSubclassOf(typeof(Module)))
            {
                var moduleInstance = (Module) Activator.CreateInstance(type);
                moduleInstance.ModuleLoader = this;
                AddModule(name, moduleInstance);
                return moduleInstance;
            }
        }
        throw new ArgumentException(string.Format("Module \"{0}\" not found", name));
    }

    public ModuleLoader LoadDirectory()
    {
        foreach (var dirname in Directory.GetDirectories(_directory))
        {
            var info = new DirectoryInfo(dirname);
            Load(info.Name, GetModuleName(info.Name));
        }
        return this;
    }

    public void LoadResources(ResourceDictionary resources)
    {
        foreach (var module in _modules.Values)
        {
            LoadModuleResources(module, resources);
        }
    }

    public void LoadModuleResources(Module module, ResourceDictionary resources)
    {
        module.LoadUserControl(resources);
    }

    public Module LoadCustomerModule(string fileName)
    {
        var name = Path.GetFileNameWithoutExtension(fileName);

        if (name == null)
        {
            throw new InvalidOperationException("Invalid module package provided");
        }
```

```
            if (_modules.ContainsKey(name))
            {
                throw new InvalidOperationException($"Module ${name} already exists");
            }
            try
            {
                ZipFile.ExtractToDirectory(fileName, Path.Combine(_directory, name));
                var module = Load(name, GetModuleName(name));
                LoadModuleResources(module, Application.Current.Resources);
                return module;
            }
            catch (Exception)
            {
                throw new InvalidOperationException("Invalid module package provided");
            }
        }
    }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<ClassDiagram MajorVersion="1" MinorVersion="1">
  <Class Name="SummarizerSharp.Data.Core.Module" Collapsed="true">
    <Position X="6.25" Y="0.75" Width="1.5" />
    <Members>
      <Property Name="ModuleLoader" Hidden="true" />
    </Members>
    <TypeIdentifier>

<HashCode>EAAAAACAEAAAEAQAAgBAAAQAAAAAAAIAgAAAAAAAAAAA=</Ha
shCode>
      <FileName>Core\Module.cs</FileName>
    </TypeIdentifier>
  </Class>
  <Class Name="SummarizerSharp.Data.ModuleBaseViewModel" Collapsed="true">
    <Position X="3" Y="0.75" Width="1.75" />
    <TypeIdentifier>

<HashCode>QRAgACMBAQQAAAAEUABQgAQEAAAAAAIChEAACAEEAFA=</Has
hCode>
      <FileName>ModuleBaseViewModel.cs</FileName>
    </TypeIdentifier>
  </Class>
  <Interface Name="SummarizerSharp.Data.Core.IPluginConfiguration" Collapsed="true">
    <Position X="4.75" Y="2" Width="1.75" />
    <TypeIdentifier>

<HashCode>AAAAAAAAAACAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</
HashCode>
      <FileName>Core\IPluginConfiguration.cs</FileName>
    </TypeIdentifier>
  </Interface>
  <Interface Name="SummarizerSharp.Data.Core.IProcessingInput" Collapsed="true">
    <Position X="6.75" Y="2" Width="1.5" />
    <TypeIdentifier>
```

```
<HashCode>AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACAAAAAIAAAQA=</
HashCode>
      <FileName>Core\IProcessingInput.cs</FileName>
    </TypeIdentifier>
  </Interface>
  <Interface Name="SummarizerSharp.Data.Core.IProcessingResult" Collapsed="true">
    <Position X="3" Y="2" Width="1.5" />
    <TypeIdentifier>

<HashCode>AAAAAAAAAAAAIAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=</
HashCode>
      <FileName>Core\IProcessingResult.cs</FileName>
    </TypeIdentifier>
  </Interface>
  <Font Name="Segoe UI" Size="9" />
</ClassDiagram>

using System.Collections.Generic;
using System.Linq;
using SharpNL.SentenceDetector;
using SharpNL.Stemmer;
using SharpNL.Tokenize;

namespace SummarizerSharp.Data.Core
{
    public static class ISentenceExtension
    {
        public static IReadOnlyList<IToken> SkipPunctuation(this ISentence sentence)
        {
            return sentence.Tokens.Where(IsPunctuation).ToList();
        }

        public static IReadOnlyList<IToken> Stemmed(this ISentence sentence, IStemmer
stemmer)
        {
            return sentence.Tokens.Select(x =>
                    new Token(x.Sentence as Sentence, x.Start, x.End,
stemmer.Stem(x.Lexeme))).ToList();
        }

        private static bool IsPunctuation(IToken token)
        {
            return !(token.Lexeme.StartsWith(".") ||
                  token.Lexeme.StartsWith(",") ||
                  token.Lexeme.StartsWith("!") ||
                  token.Lexeme.StartsWith("-") ||
                  token.Lexeme.StartsWith("?"));
        }
    }
}
```

```
using System.Collections.Generic;
using SharpNL;
using SharpNL.SentenceDetector;
using SummarizerSharp.Data.Core;

namespace SummarizerSharp.Data
{
   public class ProcessingInput: IProcessingInput
   {
      private readonly IDocument _document;

      public ProcessingInput(IDocument document, string stringData = null)
      {
         _document = document;
         StringData = stringData;
      }

      public Document Document => (Document)_document;

      public IProcessingResult ToResult()
      {
         return new ProcessingResult(_document);
      }

      public ITextFactory Factory => _document.Factory;

      public string Language => _document.Language;

      public IReadOnlyList<ISentence> Sentences
      {
         get
         {
            return _document.Sentences;
         }

         set
         {
            _document.Sentences = value;
         }
      }

      public string StringData { get; set; }

      public string Text => _document.Text;
   }
}

using System;
using System.Globalization;
using System.IO;
using SharpNL.Analyzer;
using SummarizerSharp.Data;
```

```
using SummarizerSharp.Data.Core;
using SummarizerSharp.TextLoader.Readers;
using Document = SharpNL.Document;
using TextReader = SummarizerSharp.TextLoader.Readers.Interfaces.TextReader;

namespace SummarizerSharp.TextLoader
{
   public class TextLoader : Module
   {
      private CultureInfo _lang;
      private TextReader _reader;
      private TextReaderConfig _configuration;

      public override void Configure(IPluginConfiguration config)
      {
         base.Configure(config);
         _configuration = config as TextReaderConfig;
         if (_configuration == null)
         {
            throw new ArgumentNullException();
         }
         try
         {
            var fileName = _configuration.FileName;
            var ext = Path.GetExtension(fileName);
            switch (ext)
            {
               case ".txt":
               case ".csv":
               case ".tsv":
                  _reader = new TxtTextReader(_configuration.FileName);
                  break;
               case ".doc":
               case ".docx":
                  _reader = new WordTextReader(_configuration.FileName);
                  break;
               case ".pdf":
                  _reader = new PdfTextReader(_configuration.FileName);
                  break;
               default:
                  throw new Exception();
            }
         }
         catch (Exception)
         {
            throw new InvalidOperationException("bad filename provided");
         }

         _lang = (CultureInfo)ModuleLoader.GetGlobalParameter("Culture");
      }

      public override IProcessingResult Process(IProcessingInput input)
```

```
        {
            var analyzer = new AggregateAnalyzer {
                $"{ModulePrefix}\\Models\\en-sent.bin",
                $"{ModulePrefix}\\Models\\en-token.bin"
            };

            var str = _reader.Read();
            var doc = new Document(_lang.EnglishName, str);

            analyzer.Analyze(doc);

            return new ProcessingResult(doc);
        }

        public override IPluginConfiguration GetConfiguration()
        {
            return new TextReaderConfig
            {
                FileName = ""
            };
        }

        public override Type GetViewModel()
        {
            return typeof(TextReaderViewModel);
        }

        public override Type GetUserControl()
        {
            return typeof(TextReaderUserControl);
        }
    }
}

using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Globalization;
using System.Linq;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Media;

namespace SummarizerSharp.ResultViewer
{
    public class SentenceConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo
culture)
        {
```

```
            var sent = value as ObservableCollection<MarkedSentence>;
            var textBlock = new TextBlock();
            textBlock.TextWrapping = TextWrapping.Wrap;

            if (sent == null)
            {
                return textBlock;
            }

            foreach (var sentence in sent)
            {
                var text = new Run(sentence.Text + " ");

                if (sentence.Contained)
                {
                    text.Background = new SolidColorBrush(Color.FromRgb(255, 255, 0));
                }

                textBlock.Inlines.Add(text);
            }

            return textBlock;
        }

        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo
culture)
        {
            throw new NotImplementedException();
        }
    }
}

using SharpNL.Stemmer.Snowball;
using System.Collections.Generic;
using System.Linq;

namespace SummarizerSharp.Data.Ukrainian
{
    public class UkrainianStemmer : SnowballStemmer
    {
        public static UkrainianStemmer Instance
        {
            get
            {
                if (ins == null)
                {
                    ins = new UkrainianStemmer();
                }
                return ins;
            }
        }
```

```
        static UkrainianStemmer ins = null;

        private UkrainianStemmer()
        {
           wends = word_ends.OrderByDescending(x => x.Length);
        }

        IEnumerable<string> word_ends = @"
          а ам ами ах та
          в вав вавсь вався вала валась валася вали вались валися вало валось валося вати ватись ватися всь вся
          е еві ем ею
          є ємо ємось ємося ється єте єтесь єтеся єш єшся єю
          и ив ий ила или ило илося им ими имо имось имося ите итесь итеся ити ить иться их иш ишся
          й ймо ймось ймося йсь йся йте йтесь йтеся
          і ів ій ім імо ість істю іть
          ї
          ла лась лася ло лось лося ли лись лися
          о ові овував овувала овувати ого ої ок ом ому осте ості очка очкам очками очках очки очків очкові очком очку очок ою
          ти тись тися
          у ував увала увати
          ь
          ці
          ю юст юсь юся ють ються
          я ям ями ях
        ".Trim().Split(new char[] { ' ', '\n' });

        IEnumerable<string> wends;

        IEnumerable<string> skip_ends = new List<string>() { "ер", "ск" };

        Dictionary<string, string> change_endings = new Dictionary<string, string>() {
           { "аче" , "ак" },
           { "іче" , "ік" },
           { "йовував" , "йов" }, { "йовувала", "йов" }, { "йовувати", "йов" },
           { "ьовував" , "ьов" }, { "ьовувала", "ьов" }, { "ьовувати", "ьов" },
           { "цьовував" , "ц" }, { "цьовувала", "ц" }, { "цьовувати", "ц" },
           { "ядер" , "ядр" }
        };

        // words to skip
        IEnumerable<string> stable_exclusions = new List<string> {
          "баядер", "беатріче",
          "віче",
          "наче", "неначе",
          "одначе",
          "паче"
        };

        // words to replace
```

```
Dictionary<string, string> exclusions = new Dictionary<string, string> {
    { "відер", "відр" },
    { "був", "була" }
};


public override string Stem(string word)
{
    // normalize word
    word = replaceStressedVowels(word);

    // don't change short words
    if (word.Length <= 2) return word;

    // check for unchanged exclusions
    if (stable_exclusions.Contains(word))
    {
        return word;
    }

    // check for replace exclusions
    if (exclusions.ContainsKey(word))
    {
        return exclusions[word];
    }

    // changing endings
    // TODO order endings by abc DESC
    foreach (var eow in change_endings.Keys.OrderBy(x => change_endings[x]))
    {
        if (word.EndsWith(eow))
        {
            return word.Substring(0, word.Length - eow.Length) + change_endings[eow];
        }
    }

    // match for stable endings
    foreach (var eow in skip_ends)
    {
        if (word.EndsWith(eow))
        {
            return word;
        }
    }

    // try simple trim
    foreach (var eow in wends)
    {
        if (word.EndsWith(eow))
        {
            var trimmed = word.Substring(0, word.Length - eow.Length);
            if (trimmed.Length > 2)
```

```
                {
                    return trimmed;
                }
            }
        }

        return word;
    }

    string replaceStressedVowels(string word)
    {
        Dictionary<string, string> nagolos = new Dictionary<string, string> {
            { "а́", "а" },
            { "е́", "е" },
            { "є́", "є" },
            { "и́", "и" },
            { "і́", "і" },
            { "ї́", "ї" },
            { "о́", "о" },
            { "у́", "у" },
            { "ю́", "ю" },
            { "я́", "я" }
        };

        return string.Join("", word.ToLower().Select(x =>
nagolos.ContainsKey(x.ToString()) ? nagolos[x.ToString()] : x.ToString()));
    }
  }
}

using System.Linq;
using SummarizerSharp.Data.Core;
using System.Text.RegularExpressions;

namespace SummarizerSharp.Data.Ukrainian
{
  public class UkrainianSynonymProvider : BaseSynonymProvider
  {
    const string pattern = @"";

    public UkrainianSynonymProvider(string filename): base(filename)
    { }

    protected override Synonym ParseString(string input)
    {
        var res = new Synonym();
        var split = input.Split(' ');

        res.BaseWord = split.FirstOrDefault();
        res.Synonyms = split.Skip(1)
                    .Select(x => Regex.Replace(x, @"[\,|\.]", "").ToUpper())
                    .ToList();
```

```
            return res;
        }
    }
}

using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using SharpNL;
using SharpNL.SentenceDetector;
using SharpNL.Stemmer;
using SummarizerSharp.Data;
using SummarizerSharp.Data.Core;
using SummarizerSharp.Data.Util;
using SummarizerSharp.TextRank.Util;

namespace SummarizerSharp.TextRank
{
    public class TextRank : Module
    {
        private IStemmer _stemmer;
        private SimilarityComputer _similarityComputer;
        private TextRankConfig _configuration;

        public override void Configure(IPluginConfiguration config)
        {
            _configuration = config as TextRankConfig;
            base.Configure(config);

            var language = (CultureInfo)ModuleLoader.GetGlobalParameter("Culture");
            _stemmer = StemmerFactory.Get(language);
            _similarityComputer = new SimilarityComputer(_configuration.Threads);
        }

        public override IProcessingResult Process(IProcessingInput input)
        {
            if (input == null)
            {
                throw new ArgumentException("input");
            }

            var sentences = input.Sentences
                .Select(x => x.Stemmed(_stemmer)
                              .Select(y => _stemmer.Stem(y.Lexeme)))
                .ToList();

            var matrix = _similarityComputer.Compute(sentences);

            double d = _configuration.DampingFactor;
            double[] weights = new double[sentences.Count];
```

```
            for (var iter = 0; iter < _configuration.Iterations; iter++)
            {
               for (int i = 0; i < weights.Length; i++)
               {
                  double sum = 0.0;

                  for (int j = 0; j < weights.Length; j++)
                  {
                     double local = 0.0, value = 0.0, weight = 1.0;
                     for (var k = 0; k < weights.Length; k++)
                     {
                        local += matrix[j, k];
                     }
                     if (weights[j] > 0)
                     {
                        weight = weights[j];
                     }
                     value = matrix[j, i]*weight;
                     if (local > 0)
                     {
                        value /= local;
                     }
                     sum += value;
                  }

                  weights[i] = 1 - d + d * sum;
               }
            }

            var topSentences = weights
               .Zip(input.Sentences, (x, y) => new KeyValuePair<double, ISentence>(x, y))
               .OrderByDescending(x => x.Key);

            var summary = topSentences
               .Take(Convert.ToInt32(topSentences.Count() * _configuration.Compression))
               .Select(x => x.Value)
               .OrderBy(x => x.Start)
               .Cast<Sentence>()
               .ToList();

            return new ProcessingResult(new Document(input.Language ?? input.Text,
summary));
         }

         public override IPluginConfiguration GetConfiguration()
         {
            return new TextRankConfig
            {
               Compression = 0.5,
               Iterations = 20,
               Threads = 4,
               DampingFactor = 0.85
```

```
            };
        }

        public override Type GetViewModel()
        {
            return typeof(TextRankViewModel);
        }

        public override Type GetUserControl()
        {
            return typeof(TextRankUserControl);
        }
    }
}

using System.Collections.Generic;
using SummarizerSharp.Data.Core;

namespace SummarizerSharp.Data.Tasks
{
    public class SummTaskRunAll : SummTask
    {
        public SummTaskRunAll(List<QueueItem> processingItems, string name) :
base(processingItems, name)
        {
        }

        protected override void RunInternal()
        {
            foreach (QueueItem item in ProcessingItems)
            {
                item.ModelControlViewModel.ResultStatus = null;
            }
            IProcessingInput input = null;
            var index = 1;
            foreach (var queueItem in ProcessingItems)
            {
                queueItem.Module.Configure(queueItem.Configuration);
                var metadata = new ProcessingMetadata();
                var result = queueItem.Run(input, true, metadata);
                metadata.Id = index++;
                MetadataContainer.Add(metadata);
                input = result.ToInput();
            }
        }
    }
}

<?xml version="1.0" encoding="utf-8"?>
<ClassDiagram MajorVersion="1" MinorVersion="1">
  <Class Name="SummarizerSharp.Data.Tasks.SummTask" Collapsed="true">
    <Position X="1.75" Y="0.5" Width="1.5" />
```

```
    <TypeIdentifier>

<HashCode>AEAAAAAEAAAAAAAEAAAAAAQAAAAAAAAAAAAAAAAAAAAI=</H
ashCode>
      <FileName>Tasks\SummTask.cs</FileName>
    </TypeIdentifier>
  </Class>
  <Class Name="SummarizerSharp.Data.Tasks.SummTaskFromIndex" Collapsed="true">
    <Position X="0.75" Y="2" Width="1.75" />
    <TypeIdentifier>

<HashCode>AAAAAAAAAAAAAAAAEAAAgAAAAAAAAAAAAAAAAAAAAAA=</
HashCode>
      <FileName>Tasks\SummTaskFromIndex.cs</FileName>
    </TypeIdentifier>
  </Class>
  <Class Name="SummarizerSharp.Data.Tasks.SummTaskRunAll" Collapsed="true">
    <Position X="2.75" Y="2" Width="1.5" />
    <TypeIdentifier>

<HashCode>AAAAAAAAAAAAAAAAEAAAAAAAAAAAAAAAAAAAAAAAAAA=</
HashCode>
      <FileName>Tasks\SummTaskRunAll.cs</FileName>
    </TypeIdentifier>
  </Class>
  <Enum Name="SummarizerSharp.Data.Tasks.ProcessingStatus">
    <Position X="4.5" Y="0.5" Width="1.5" />
    <TypeIdentifier>

<HashCode>AAAAAAAAAAAAAQAAAAQAAAAAAAAAAAIAAAAIAAAAAA=</H
ashCode>
      <FileName>Tasks\SummTask.cs</FileName>
    </TypeIdentifier>
  </Enum>
  <Font Name="Segoe UI" Size="9" />
</ClassDiagram>

using System;
using System.Collections.ObjectModel;
using System.Linq;
using System.Threading.Tasks;

namespace SummarizerSharp.Data.Tasks
{
    public class TaskManager : BaseViewModel
    {
        private static readonly Lazy<TaskManager> _instance = new Lazy<TaskManager>(() =>
new TaskManager());
        private readonly object _lockObj = new object();

        private TaskManager()
        {
```

```
            Task.Run(() => Run());
        }

        public static TaskManager Instance => _instance.Value;

        public static int EmptyPendingTasksDelay { get; set; } = 500;

        public ObservableCollection<SummTask> InputQueue { get; } = new
ObservableCollection<SummTask>();

        private async void Run()
        {
            while (true)
            {
                SummTask item;
                lock (_lockObj)
                {
                    item = InputQueue.FirstOrDefault(x => x.Status == ProcessingStatus.Pending);
                }

                if (item == null)
                {
                    await Task.Delay(EmptyPendingTasksDelay);
                    continue;
                }

                await item.Run();
            }
        }

        public void AddTask(SummTask task)
        {
            task.Status = ProcessingStatus.Pending;
            lock (_lockObj)
            {
                InputQueue.Add(task);
            }
        }
    }
}
```